Aerobus

v1.2

Generated by Doxygen 1.9.8

Chapter 1

Introduction

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

Everything in Aerobus is expressed as types.

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

Everything is expressed as types

The library serves two main purposes :

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

It is designed to be 'quite easily' extensible.

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

1.1 HOW TO

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

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

1.1.1 Unit Test

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

Move to the top directory then :
```
cmake -S . -B build
cmake --build build
cd build && ctest
```

Terminal should write :
```
100% tests passed, 0 tests failed out of 48
```

Alternate way :
```
make tests
```

From top directory.

1.1.2 Benchmarks

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

In addition of Cmake and compiler, install OpenMP. And Google's Benchmark library. Then move to top directory :
```
rm -rf build
mkdir build
cd build
cmake ..
make benchmarks
./benchmarks
```

1.2 Structures

1.2.1 Predefined discrete euclidean domains

Aerobus predefines several simple euclidean domains, such as :

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

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

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

and the following "elements" :

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

1.2.2 Polynomials

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

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

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

1.2.3 Known polynomials

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

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

- `chebyshev_T`
- `chebyshev_U`
- `laguerre`
- `hermite_prob`
- `hermite_phys`
- `bernstein`
- `legendre`
- `bernoulli`

1.2.4 Conway polynomials

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

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

1.2.5 Taylor series

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

They can be used and evaluated:

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

Exposed functions are:

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

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

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

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

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

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

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

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

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

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

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

```
compute_expm1(unsigned long, double const*, double*):
  lea     rax, [rdi-1]
  cmp     rax, 2
  jbe     .L5
  mov     rcx, rdi
  xor     eax, eax
  vxorpd  xmm1, xmm1, xmm1
  vbroadcastsd    ymm14, QWORD PTR .LC1[rip]
  vbroadcastsd    ymm13, QWORD PTR .LC3[rip]
  shr     rcx, 2
  vbroadcastsd    ymm12, QWORD PTR .LC5[rip]
  vbroadcastsd    ymm11, QWORD PTR .LC7[rip]
  sal     rcx, 5
  vbroadcastsd    ymm10, QWORD PTR .LC9[rip]
  vbroadcastsd    ymm9, QWORD PTR .LC11[rip]
  vbroadcastsd    ymm8, QWORD PTR .LC13[rip]
  vbroadcastsd    ymm7, QWORD PTR .LC15[rip]
  vbroadcastsd    ymm6, QWORD PTR .LC17[rip]
  vbroadcastsd    ymm5, QWORD PTR .LC19[rip]
  vbroadcastsd    ymm4, QWORD PTR .LC21[rip]
  vbroadcastsd    ymm3, QWORD PTR .LC23[rip]
  vbroadcastsd    ymm2, QWORD PTR .LC25[rip]
.L3:
  vmovupd ymm15, YMMWORD PTR [rsi+rax]
  vmovapd ymm0, ymm15
  vfmadd132pd     ymm0, ymm14, ymm1
  vfmadd132pd     ymm0, ymm13, ymm15
  vfmadd132pd     ymm0, ymm12, ymm15
  vfmadd132pd     ymm0, ymm11, ymm15
  vfmadd132pd     ymm0, ymm10, ymm15
  vfmadd132pd     ymm0, ymm9, ymm15
  vfmadd132pd     ymm0, ymm8, ymm15
  vfmadd132pd     ymm0, ymm7, ymm15
  vfmadd132pd     ymm0, ymm6, ymm15
  vfmadd132pd     ymm0, ymm5, ymm15
  vfmadd132pd     ymm0, ymm4, ymm15
  vfmadd132pd     ymm0, ymm3, ymm15
  vfmadd132pd     ymm0, ymm2, ymm15
  vfmadd132pd     ymm0, ymm1, ymm15
  vmovupd YMMWORD PTR [rdx+rax], ymm0
  add     rax, 32
  cmp     rcx, rax
  jne     .L3
  mov     rax, rdi
  and     rax, -4
  vzeroupper
```

1.3 Operations

1.3.1 Field of fractions

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

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

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

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

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

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

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

Which also have an evaluation function, as polynomial do.

1.3.2 Quotient

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

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

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

As we could have using zpz<17>.

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

1.4 Misc

1.4.1 Continued Fractions

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

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

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

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

1.5 CUDA

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

Unfortunately, NVIDIA did not put enough constexpr in its `cuda_fp16.h` header, so we had to implement our own constexpr static_cast from int16_t to `__half` to make integers polynomials work with `__half`. See this bug.

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

A workaround is to modify `cuda_fp16.h` and add a constexpr modifier to line 5039. This works but only tested on Linux with CUDA 16.1.

Once done, nvcc generates splendid assembly, same as for `double` or `float`:

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

Please push to make these bug fixed by NVIDIA.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Concept Index

3.1 Concepts

Here is a list of all concepts with brief descriptions:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 aerobus Namespace Reference

main namespace for all publicly exposed types or functions

Namespaces

- namespace internal

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

  families of well known polynomials such as Hermite or Bernstein
- namespace libm

Classes

- struct ContinuedFraction

  represents a continued fraction a0 + $\frac{1}{a_1+\frac{1}{a_2+\dots}}$
- struct ContinuedFraction< a0 >

  Specialization for only one coefficient, technically just 'a0'.
- struct ContinuedFraction< a0, rest... >

  specialization for multiple coefficients (strictly more than one)
- struct ConwayPolynomial
- struct Embed

  embedding - struct forward declaration
- struct Embed< i32, i64 >

  embeds i32 into i64
- struct Embed< polynomial< Small >, polynomial< Large > >

  embeds polynomial<Small> into polynomial<Large>
- struct Embed< q32, q64 >

  embeds q32 into q64
- struct Embed< Quotient< Ring, X >, Ring >

  embeds Quotient<Ring, X> into Ring
- struct Embed< Ring, FractionField< Ring > >

  embeds values from Ring to its field of fractions

- struct Embed< zpz< x >, i32 >

  embeds zpz values into i32
- struct i32

  32 bits signed integers, seen as a algebraic ring with related operations
- struct i64

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

  checks if n is prime
- struct meta_libm
- struct polynomial
- struct Quotient

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

  Empty pure template struct to handle type list.
- struct type_list<>

  specialization for empty type list
- struct zpz

  congruence classes of integers modulo p (32 bits)

Concepts

- concept IsRing

  Concept to express R is a Ring.
- concept IsEuclideanDomain

  Concept to express R is an euclidean domain.
- concept IsField

  Concept to express R is a field.

Typedefs

- template<typename T , typename A , typename B >
  using gcd_t = typename internal::gcd< T >::template type< A, B >

  computes the greatest common divisor or A and B
- template<typename... vals>
  using vadd_t = typename internal::vadd< vals... >::type

  adds multiple values (v1 + v2 + ... + vn) vals must have same "enclosing_type" and "enclosing_type" must have an add_t binary operator
- template<typename... vals>
  using vmul_t = typename internal::vmul< vals... >::type

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

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

  Fraction field of an euclidean domain, such as Q for Z.
- template<typename X , typename Y >
  using add_t = typename X::enclosing_type::template add_t< X, Y >

  generic addition

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

  generic subtraction
- template<typename X , typename Y >
  using mul_t = typename X::enclosing_type::template mul_t< X, Y >

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

  generic division
- using q32 = FractionField< i32 >

  32 bits rationals rationals with 32 bits numerator and denominator
- using fpq32 = FractionField< polynomial< q32 > >

  rational fractions with 32 bits rational coefficients rational fractions with rationals coefficients (32 bits numerator and denominator)
- using q64 = FractionField< i64 >

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

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

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

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

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

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

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

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

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

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

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

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

computes factorial(i), as type

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

  computes binomial coefficient (k among n) as type
- template<typename T , size_t n>
  using bernoulli_t = typename internal::bernoulli< T, n >::type

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

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

  (-1)^k as type in T
- template<typename T , int n, int k>
  using stirling_1_signed_t = typename internal::stirling_1_helper< T, n, k >::type

  Stirling number of first king (signed) – as types.
- template<typename T , int n, int k>
  using stirling_1_unsigned_t = abs_t< typename internal::stirling_1_helper< T, n, k >::type >

  Stirling number of first king (unsigned) – as types.
- template<typename T , int n, int k>
  using stirling_2_t = typename internal::stirling_2_helper< T, n, k >::type

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

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

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

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

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

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

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

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

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

  $\cos(x)$ *cosinus*

- template<typename Integers , size_t deg>
  using geometric_sum = taylor< Integers, internal::geom_coeff, deg >
  $\frac{1}{1-x}$ *zero development of* $\frac{1}{1-x}$
- template<typename Integers , size_t deg>
  using asin = taylor< Integers, internal::asin_coeff, deg >
  $\arcsin(x)$ *arc sinus*
- template<typename Integers , size_t deg>
  using asinh = taylor< Integers, internal::asinh_coeff, deg >
  $\operatorname{arcsinh}(x)$ *arc hyperbolic sinus*
- template<typename Integers , size_t deg>
  using atanh = taylor< Integers, internal::atanh_coeff, deg >
  $\operatorname{arctanh}(x)$ *arc hyperbolic tangent*
- template<typename Integers , size_t deg>
  using tan = taylor< Integers, internal::tan_coeff, deg >
  $\tan(x)$ *tangent*
- template<typename Integers , size_t deg>
  using tanh = taylor< Integers, internal::tanh_coeff, deg >
  $\tanh(x)$ *hyperbolic tangent*
- using PI_fraction = ContinuedFraction< 3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1 >
- using E_fraction = ContinuedFraction< 2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1 >
  approximation of e
- using SQRT2_fraction = ContinuedFraction< 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 >
  approximation of $\sqrt{2}$
- using SQRT3_fraction = ContinuedFraction< 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 >
  approximation of

Functions

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

Variables

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

6.1.1 Detailed Description

main namespace for all publicly exposed types or functions

6.1.2 Typedef Documentation

6.1.2.1 abs_t

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

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

Template Parameters

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

6.1.2.2 add_t

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

generic addition

Template Parameters

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

6.1.2.3 addfractions_t

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

helper type : adds two fractions

Template Parameters

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

6.1.2.4 alternate_t

```
template<typename T , int k>
using aerobus::alternate_t = typedef typename internal::alternate<T, k>::type
```

(-1)^k as type in T

Template Parameters

T	Ring type, aerobus::i64 for example

6.1.2.5 asin

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

$\arcsin(x)$ arc sinus

Template Parameters

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

6.1.2.6 asinh

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

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

Template Parameters

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

6.1.2.7 atan

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

$\arctan(x)$

Template Parameters

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

6.1.2.8 atanh

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

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

Template Parameters

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

6.1.2.9 bell_t

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

Bell numbers.

Template Parameters

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

6.1.2.10 bernoulli_t

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

nth bernoulli number as type in T

Template Parameters

T	Ring type (i64)
n	

6.1.2.11 combination_t

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

computes binomial coefficient (k among n) as type

Template Parameters

T	Ring type (i32 for example)

6.1.2.12 cos

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

$\cos(x)$ cosinus

Template Parameters

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

6.1.2.13 cosh

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

$\cosh(x)$ hyperbolic cosine

Template Parameters

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

6.1.2.14 div_t

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

generic division

Template Parameters

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

6.1.2.15 E_fraction

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

approximation of e

6.1.2.16 embed_int_poly_in_fractions_t

```
template<typename v >
using aerobus::embed_int_poly_in_fractions_t = typedef typename Embed< polynomial<typename v↩
::ring_type>, polynomial<FractionField<typename v::ring_type> >>::template type<v>
```

embed a polynomial with integers coefficients into rational coefficients polynomials

Lives in polynomial<FractionField<Ring>>

Template Parameters

Ring	Integers
a	value in polynomial<Ring>

6.1.2.17 exp

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

e^x

Template Parameters

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

6.1.2.18 expm1

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

$e^x - 1$

Template Parameters

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

6.1.2.19 factorial_t

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

computes factorial(i), as type

Template Parameters

T	Ring type (e.g. i32)
i	

6.1.2.20 fpq32

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

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

6.1.2.21 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.22 FractionField

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

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

Template Parameters

Ring	

6.1.2.23 gcd_t

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

computes the greatest common divisor or A and B

Template Parameters

T	Ring type (must be euclidean domain)

6.1.2.24 geometric_sum

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

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

Template Parameters

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

6.1.2.25 lnp1

```
template<typename Integers , size_t deg>
using aerobus::lnp1 = typedef taylor<Integers, internal::lnp1_coeff, deg>
```

$\ln(1+x)$

Template Parameters

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

6.1.2.26 make_frac_polynomial_t

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

make a polynomial with coefficients in FractionField<Ring>

Template Parameters

Ring	integers
...xs	values

6.1.2.27 make_int_polynomial_t

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

make a polynomial with coefficients in Ring

Template Parameters

Ring	integers
...xs	coefficients

6.1.2.28 make_q32_t

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

helper type : make a fraction from numerator and denominator

Template Parameters

p	numerator
q	denominator

6.1.2.29 make_q64_t

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

helper type : make a fraction from numerator and denominator

Template Parameters

p	numerator
q	denominator

6.1.2.30 makefraction_t

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

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

Template Parameters

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

6.1.2.31 mul_t

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

generic multiplication

Template Parameters

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

6.1.2.32 mulfractions_t

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

helper type : multiplies two fractions

Template Parameters

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

6.1.2.33 pi64

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

polynomial with 64 bits integers coefficients

6.1.2.34 PI_fraction

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

representation of π as a continued fraction

6.1.2.35 pow_t

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

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

Template Parameters

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

6.1.2.36 pq64

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

polynomial with 64 bits rationals coefficients

6.1.2.37 q32

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

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.38 q64

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

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.39 sin

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

$\sin(x)$

Template Parameters

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

6.1.2.40 sinh

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

$\sinh(x)$

Template Parameters

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

6.1.2.41 SQRT2_fraction

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

approximation of $\sqrt{2}$

6.1.2.42 SQRT3_fraction

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

approximation of

6.1.2.43 stirling_1_signed_t

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

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

Template Parameters

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

6.1.2.44 stirling_1_unsigned_t

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

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

Template Parameters

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

6.1.2.45 stirling_2_t

```
template<typename T , int n, int k>
using aerobus::stirling_2_t = typedef typename internal::stirling_2_helper<T, n, k>::type
```

Stirling number of second king – as types.

Template Parameters

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

6.1.2.46 sub_t

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

generic subtraction

Template Parameters

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

6.1.2.47 tan

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

$\tan(x)$ tangent

Template Parameters

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

6.1.2.48 tanh

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

$\tanh(x)$ hyperbolic tangent

Template Parameters

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

6.1.2.49 taylor

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

Template Parameters

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

6.1.2.50 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	

6.1.2.51 vmul_t

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

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

Template Parameters

...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 **AbelHelper**
- struct **AllOneHelper**
- struct **AllOneHelper**< **0, I** >
- struct **alternate**
- struct **alternate**< **T, k, std::enable_if_t**< **k % 2 !=0** > >
- struct **alternate**< **T, k, std::enable_if_t**< **k % 2==0** > >
- struct **arithmetic_helpers**
- struct **arithmetic_helpers**< **double** >
- struct **arithmetic_helpers**< **float** >
- struct **asin_coeff**
- struct **asin_coeff_helper**
- struct **asin_coeff_helper**< **T, i, std::enable_if_t**<(**i &1)==0** > >
- struct **asin_coeff_helper**< **T, i, std::enable_if_t**<(**i &1)==1** > >
- struct **asinh_coeff**

- struct **asinh_coeff_helper**
- struct **asinh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **asinh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **atan_coeff**
- struct **atan_coeff_helper**
- struct **atan_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **atan_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **atanh_coeff**
- struct **atanh_coeff_helper**
- struct **atanh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **atanh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **bell_helper**
- struct **bell_helper< T, 0 >**
- struct **bell_helper< T, 1 >**
- struct **bell_helper< T, n, std::enable_if_t<(n > 1)> >**
- struct **bernoulli**
- struct **bernoulli< T, 0 >**
- struct **bernoulli_coeff**
- struct **bernoulli_helper**
- struct **bernoulli_helper< T, accum, m, m >**
- struct **bernstein_helper**
- struct **bernstein_helper< 0, 0, l >**
- struct **bernstein_helper< i, m, l, std::enable_if_t<(m > 0) &&(i > 0) &&(i< m)> >**
- struct **bernstein_helper< i, m, l, std::enable_if_t<(m > 0) &&(i==0)> >**
- struct **bernstein_helper< i, m, l, std::enable_if_t<(m > 0) &&(i==m)> >**
- struct **BesselHelper**
- struct **BesselHelper< 0, l >**
- struct **BesselHelper< 1, l >**
- struct **chebyshev_helper**
- struct **chebyshev_helper< 1, 0, l >**
- struct **chebyshev_helper< 1, 1, l >**
- struct **chebyshev_helper< 2, 0, l >**
- struct **chebyshev_helper< 2, 1, l >**
- struct **combination**
- struct **combination_helper**
- struct **combination_helper< T, 0, n >**
- struct **combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k >(n/2) &&k > 0)> >**
- struct **combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k<=(n/2) &&k > 0)> >**
- struct **cos_coeff**
- struct **cos_coeff_helper**
- struct **cos_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **cos_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **cosh_coeff**
- struct **cosh_coeff_helper**
- struct **cosh_coeff_helper< T, i, std::enable_if_t<(i &1)==0 > >**
- struct **cosh_coeff_helper< T, i, std::enable_if_t<(i &1)==1 > >**
- struct **double_double**
- struct **exp_coeff**
- struct **factorial**
- struct **factorial< T, 0 >**
- struct **factorial< T, x, std::enable_if_t<(x > 0)> >**
- struct **FloatLayout**
- struct **FloatLayout< double >**
- struct **FloatLayout< float >**
- struct **FloatLayout< long double >**

- struct **fma_helper**
- struct **fma_helper**< **double** >
- struct **fma_helper**< **float** >
- struct **fma_helper**< **int16_t** >
- struct **fma_helper**< **int32_t** >
- struct **fma_helper**< **int64_t** >
- struct **fma_helper**< **long double** >
- struct **FractionFieldImpl**
- struct **FractionFieldImpl**< **Field, std::enable_if_t**< **Field::is_field** > >
- struct **FractionFieldImpl**< **Ring, std::enable_if_t**<**!Ring::is_field** > >
- struct **gcd**

  greatest common divisor computes the greatest common divisor exposes it in gcd<A, B>::type as long as Ring type is an integral domain
- struct **gcd**< **Ring, std::enable_if_t**< **Ring::is_euclidean_domain** > >
- struct **geom_coeff**
- struct **hermite_helper**
- struct **hermite_helper**< **0, known_polynomials::hermite_kind::physicist, I** >
- struct **hermite_helper**< **0, known_polynomials::hermite_kind::probabilist, I** >
- struct **hermite_helper**< **1, known_polynomials::hermite_kind::physicist, I** >
- struct **hermite_helper**< **1, known_polynomials::hermite_kind::probabilist, I** >
- struct **hermite_helper**< **deg, known_polynomials::hermite_kind::physicist, I** >
- struct **hermite_helper**< **deg, known_polynomials::hermite_kind::probabilist, I** >
- struct **insert_h**
- struct **is_instantiation_of**
- struct **is_instantiation_of**< **TT, TT**< **Ts...** > >
- struct **laguerre_helper**
- struct **laguerre_helper**< **0, I** >
- struct **laguerre_helper**< **1, I** >
- struct **legendre_helper**
- struct **legendre_helper**< **0, I** >
- struct **legendre_helper**< **1, I** >
- struct **lnp1_coeff**
- struct **lnp1_coeff**< **T, 0** >
- struct **make_taylor_impl**
- struct **make_taylor_impl**< **T, coeff_at, std::integer_sequence**< **size_t, Is...** > >
- struct **pop_front_h**
- struct **pow**
- struct **pow**< **T, p, n, std::enable_if_t**< **n==0** > >
- struct **pow**< **T, p, n, std::enable_if_t**<**(n % 2==1)**> >
- struct **pow**< **T, p, n, std::enable_if_t**<**(n > 0 &&n % 2==0)**> >
- struct **pow_scalar**
- struct **remove_h**
- struct **sh_coeff**
- struct **sh_coeff_helper**
- struct **sh_coeff_helper**< **T, i, std::enable_if_t**<**(i &1)==0** > >
- struct **sh_coeff_helper**< **T, i, std::enable_if_t**<**(i &1)==1** > >
- struct **sin_coeff**
- struct **sin_coeff_helper**
- struct **sin_coeff_helper**< **T, i, std::enable_if_t**<**(i &1)==0** > >
- struct **sin_coeff_helper**< **T, i, std::enable_if_t**<**(i &1)==1** > >
- struct **Split**
- struct **split_h**
- struct **split_h**< **0, L1, L2** >
- struct **staticcast**

- struct **stirling_1_helper**
- struct **stirling_1_helper< T, 0, 0 >**
- struct **stirling_1_helper< T, 0, n, std::enable_if_t<(n > 0)> >**
- struct **stirling_1_helper< T, n, 0, std::enable_if_t<(n > 0)> >**
- struct **stirling_1_helper< T, n, k, std::enable_if_t<(k > 0) &&(n > 0)> >**
- struct **stirling_2_helper**
- struct **stirling_2_helper< T, 0, n, std::enable_if_t<(n > 0)> >**
- struct **stirling_2_helper< T, n, 0, std::enable_if_t<(n > 0)> >**
- struct **stirling_2_helper< T, n, k, std::enable_if_t<(k > 0) &&(n > 0) &&(k< n)> >**
- struct **stirling_2_helper< T, n, n, std::enable_if_t<(n >=0)> >**
- struct **tan_coeff**
- struct **tan_coeff_helper**
- struct **tan_coeff_helper< T, i, std::enable_if_t<(i % 2) !=0 > >**
- struct **tan_coeff_helper< T, i, std::enable_if_t<(i % 2)==0 > >**
- struct **tanh_coeff**
- struct **tanh_coeff_helper**
- struct **tanh_coeff_helper< T, i, std::enable_if_t<(i % 2) !=0 > >**
- struct **tanh_coeff_helper< T, i, std::enable_if_t<(i % 2)==0 > >**
- struct **touchard_coeff**
- struct **type_at**
- struct **type_at< 0, T, Ts... >**
- struct **vadd**
- struct **vadd< v1 >**
- struct **vadd< v1, vals... >**
- struct **vmul**
- struct **vmul< v1 >**
- struct **vmul< v1, vals... >**

Typedefs

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

Functions

- template<std::size_t... 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

Enumerations

- enum hermite_kind { probabilist , physicist }

6.3.1 Detailed Description

families of well known polynomials such as Hermite or Bernstein

6.3.2 Enumeration Type Documentation

6.3.2.1 hermite_kind

```
enum aerobus::known_polynomials::hermite_kind
```

Enumerator

probabilist	
physicist	

6.4 aerobus::libm Namespace Reference

Namespaces

- namespace internal

Functions

- brief computes x (NOT TESTED YET) @tparam T @param x @return template< typename T > static T exp2(const T &x)
- brief computes sine following IEEE recommendations exact values for and correct at epsilon for other values tparam T arithmetic_type (float only so far) @param x input value @return sin(x) template< typename T > static T sin(const T &x)

Variables

- brief computes sine function
- brief computes sine following IEEE recommendations exact values for and infinities

6.4.1 Function Documentation

6.4.1.1 arithmetic_type()

```
brief computes cosine following IEEE recommendations exact values for and correct at epsilon
for other values tparam T aerobus::libm::arithmetic_type (
            float only so far ) const &
```

6.4.1.2 x()

```
brief computes aerobus::libm::x (
            NOT TESTED YET ) const &
```

6.4.2 Variable Documentation

6.4.2.1 function

```
brief computes cosine aerobus::libm::function
```

6.4.2.2 infinities

```
brief computes cosine following IEEE recommendations exact values for and aerobus::libm↩
::infinities
```

6.5 aerobus::libm::internal Namespace Reference

Classes

- struct cos_poly
- struct cos_poly< double >
- struct cos_poly< float >
- struct exp2_poly
- struct exp2_poly< double >
- struct exp2_poly< float >
- struct sin_poly
- struct sin_poly< double >
- struct sin_poly< float >

Chapter 7

Concept Documentation

7.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

`#include <aerobus.h>`

7.1.1 Concept definition

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

        R::template pos_v<typename R::one> == true;

        R::is_euclidean_domain == true;
    }
```

7.1.2 Detailed Description

Concept to express R is an euclidean domain.

7.2 aerobus::IsField Concept Reference

Concept to express R is a field.

`#include <aerobus.h>`

7.2.1 Concept definition

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

7.2.2 Detailed Description

Concept to express R is a field.

7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

`#include <aerobus.h>`

7.3.1 Concept definition

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

7.3.2 Detailed Description

Concept to express R is a Ring.

Chapter 8

Class Documentation

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

`#include <aerobus.h>`

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

- src/aerobus.h

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

`#include <aerobus.h>`

Public Types

- using type = typename Ring::zero

8.2.1 Member Typedef Documentation

8.2.1.1 type

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

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

- src/aerobus.h

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

`#include <aerobus.h>`

Public Types

- using type = aN

8.3.1 Member Typedef Documentation

8.3.1.1 type

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

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

- src/aerobus.h

8.4 aerobus::ContinuedFraction< values > Struct Template Reference

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

`#include <aerobus.h>`

8.4.1 Detailed Description

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

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

Template Parameters

...values	are int64_t

Examples

examples/continued_fractions.cpp.

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

- src/aerobus.h

8.5 aerobus::ContinuedFraction< a0 > Struct Template Reference

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

`#include <aerobus.h>`

Public Types

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

Static Public Attributes

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

8.5.1 Detailed Description

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

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

Template Parameters

a0	an integer int64_t

8.5.2 Member Typedef Documentation

8.5.2.1 type

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

represented value as aerobus::q64

8.5.3 Member Data Documentation

8.5.3.1 val

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

represented value as double

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

- src/aerobus.h

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

specialization for multiple coefficients (strictly more than one)

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

Public Types

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

  represented value as aerobus::q64

Static Public Attributes

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

  reprensented value as double

8.6.1 Detailed Description

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

specialization for multiple coefficients (strictly more than one)

Template Parameters

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

8.6.2 Member Typedef Documentation

8.6.2.1 type

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

represented value as aerobus::q64

8.6.3 Member Data Documentation

8.6.3.1 val

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

reprensented value as double

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

- src/aerobus.h

8.7 aerobus::ConwayPolynomial Struct Reference

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

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

- src/aerobus.h

8.8 aerobus::libm::internal::cos_poly< T > Struct Template Reference

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

- src/aerobus.h

8.9 aerobus::libm::internal::cos_poly< double > Struct Reference

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

Public Types

- using type = typename aerobus::polynomial< aerobus::q64 >::simplify_t< typename aerobus::polynomial< aerobus::q64 >::template val< aerobus::make_q64_t< 14, 407833194230757 >, aerobus::make_q64_t<-283, 24728864074278 >, aerobus::make_q64_t< 136, 65144855767 >, aerobus::make_q64_t<-4089, 14838163607 >, aerobus::make_q64_t< 452396, 18240606721 >, aerobus::make_q64_t<-6405119470037555, 4611686018427387904 >, aerobus::make_q64_t< 6004799503160661, 144115188075855872 > > >

8.9.1 Member Typedef Documentation

8.9.1.1 type

```
using aerobus::libm::internal::cos_poly< double >::type = typename aerobus::polynomial<aerobus::q64>↩
::simplify_t< typename aerobus::polynomial<aerobus::q64>::  template val< aerobus::make_q64_t<14,
407833194230757>, aerobus::make_q64_t<-283, 24728864074278>, aerobus::make_q64_t<136, 65144855767>,
aerobus::make_q64_t<-4089, 14838163607>, aerobus::make_q64_t<452396, 18240606721>, aerobus::make_q64_t<-6405
4611686018427387904>, aerobus::make_q64_t<6004799503160661, 144115188075855872> >>
```

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

- src/aerobus.h

8.10 aerobus::libm::internal::cos_poly< float > Struct Reference

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

Public Types

- using type = typename aerobus::polynomial< aerobus::q32 >::simplify_t< typename aerobus::polynomial< aerobus::q32 >::template val< aerobus::make_q32_t<-1, 3112816 >, aerobus::make_q32_t< 1, 40237 >, aerobus::make_q32_t<-119, 85679 >, aerobus::make_q32_t< 11184811, 268435456 > > >

8.10.1 Member Typedef Documentation

8.10.1.1 type

```
using aerobus::libm::internal::cos_poly< float >::type = typename aerobus::polynomial<aerobus::q32>↩
::simplify_t< typename aerobus::polynomial<aerobus::q32>::  template val< aerobus::make_q32_t<-1,
3112816>, aerobus::make_q32_t<1, 40237>, aerobus::make_q32_t<-119, 85679>, aerobus::make_q32_t<11184811,
268435456> >>
```

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

- src/aerobus.h

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

`#include <aerobus.h>`

Static Public Member Functions

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

8.11.1 Member Function Documentation

8.11.1.1 func()

```
template<typename Ring >
template<typename arithmeticType , typename P >
template<int64_t index, int ghost>
static INLINED DEVICE void aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P
>::EFTHorner< index, ghost >::func (
             arithmeticType x,
             arithmeticType * pi,
             arithmeticType * sigma,
             arithmeticType * r )  [inline], [static]
```

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

- src/aerobus.h

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

`#include <aerobus.h>`

Static Public Member Functions

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

8.12.1 Member Function Documentation

8.12.1.1 func()

```
template<typename Ring >
template<typename arithmeticType , typename P >
template<int ghost>
static INLINED DEVICE void aerobus::polynomial< Ring >::compensated_horner< arithmeticType, P
>::EFTHorner<-1, ghost >::func (
             arithmeticType x,
             arithmeticType * pi,
             arithmeticType * sigma,
             arithmeticType * r )  [inline], [static]
```

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

- src/aerobus.h

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

embedding - struct forward declaration

8.13.1 Detailed Description

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

embedding - struct forward declaration

Template Parameters

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

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

- src/aerobus.h

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

embeds i32 into i64

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

Public Types

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

8.14.1 Detailed Description

embeds i32 into i64

8.14.2 Member Typedef Documentation

8.14.2.1 type

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

the i64 representation of val

Template Parameters

val	a value in i32

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

- src/aerobus.h

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

embeds polynomial<Small> into polynomial<Large>

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

Public Types

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

8.15.1 Detailed Description

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

embeds polynomial<Small> into polynomial<Large>

Template Parameters

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

8.15.2 Member Typedef Documentation

8.15.2.1 type

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

the polynomial<Large> reprensentation of v

Template Parameters

v	a value in polynomial<Small>

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

- src/aerobus.h

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

embeds q32 into q64

`#include <aerobus.h>`

Public Types

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

8.16.1 Detailed Description

embeds q32 into q64

8.16.2 Member Typedef Documentation

8.16.2.1 type

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

q64 representation of v

Template Parameters

v	a value in q32

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

- src/aerobus.h

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

embeds Quotient<Ring, X> into Ring

`#include <aerobus.h>`

Public Types

- template<typename val >
  using type = typename val::raw_t

  Ring reprensentation of val.

8.17.1 Detailed Description

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

embeds Quotient<Ring, X> into Ring

Template Parameters

Ring	a Euclidean ring
X	a value in Ring

8.17.2 Member Typedef Documentation

8.17.2.1 type

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

Ring reprensentation of val.

Template Parameters

val	a value in Quotient<Ring, X>

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

- src/aerobus.h

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

embeds values from Ring to its field of fractions

`#include <aerobus.h>`

Public Types

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

8.18.1 Detailed Description

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

embeds values from Ring to its field of fractions

Template Parameters

Ring	an integers ring, such as i32

8.18.2 Member Typedef Documentation

8.18.2.1 type

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

FractionField<Ring> reprensentation of v.

Template Parameters

v	a Ring value

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

- src/aerobus.h

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

embeds zpz values into i32

`#include <aerobus.h>`

Public Types

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

8.19.1 Detailed Description

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

embeds zpz values into i32

Template Parameters

x	an integer

8.19.2 Member Typedef Documentation

8.19.2.1 type

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

the i32 reprensentation of val

Template Parameters

val	a value in zpz<x>

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

- src/aerobus.h

8.20 aerobus::libm::internal::exp2_poly< T > Struct Template Reference

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

- src/aerobus.h

8.21 aerobus::libm::internal::exp2_poly< double > Struct Reference

`#include <aerobus.h>`

Public Types

- using type = aerobus::polynomial< aerobus::q64 >::template val< aerobus::make_q64_t< 388, 10641171255427 >, aerobus::make_q64_t< 2296, 5579977897299 >, aerobus::make_q64_t< 4963, 697799188641 >, aerobus::make_q64_t< 15551, 152884735543 >, aerobus::make_q64_t< 21273, 16096444733 >, aerobus::make_q64_t< 683500, 44811710339 >, aerobus::make_q64_t< 44397656049575, 288230376151711744 >, aerobus::make_q64_t< 192156823709857, 144115188075855872 >, aerobus::make_q64_t< 693059242663871, 72057594037927936 >, aerobus::make_q64_t< 1999746264802375, 360287970189639 68 >, aerobus::make_q64_t< 4327536028902111, 18014398509481984 >, aerobus::make_q64_t< 6243314768165359, 9007199254740992 >, aerobus::q64::one >

8.21.1 Member Typedef Documentation

8.21.1.1 type

```
using aerobus::libm::internal::exp2_poly< double >::type = aerobus::polynomial<aerobus::q64>↩
::template val< aerobus::make_q64_t<388, 10641171255427>, aerobus::make_q64_t<2296, 5579977897299>,
aerobus::make_q64_t<4963, 697799188641>, aerobus::make_q64_t<15551, 152884735543>, aerobus::make_q64_t<21273
16096444733>, aerobus::make_q64_t<683500, 44811710339>, aerobus::make_q64_t<44397656049575,
288230376151711744>, aerobus::make_q64_t<192156823709857, 144115188075855872>, aerobus::make_q64_t<693059242
72057594037927936>, aerobus::make_q64_t<1999746264802375, 360287970189639 68>, aerobus::make_q64_t<4327536028
18014398509481984>, aerobus::make_q64_t<6243314768165359, 9007199254740992>, aerobus::q64↩
::one>
```

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

- src/aerobus.h

8.22 aerobus::libm::internal::exp2_poly< float > Struct Reference

`#include <aerobus.h>`

Public Types

- using type = aerobus::polynomial< aerobus::q32 >::template val< aerobus::make_q32_t< 8, 375115 >, aerobus::make_q32_t< 30, 208117 >, aerobus::make_q32_t< 109, 81261 >, aerobus::make_q32_t< 5161841, 536870912 >, aerobus::make_q32_t< 3724869, 67108864 >, aerobus::make_q32_t< 16121323, 67108864 >, aerobus::make_q32_t< 1453635, 2097152 >, aerobus::q32::one >

8.22.1 Member Typedef Documentation

8.22.1.1 type

```
using aerobus::libm::internal::exp2_poly< float >::type = aerobus::polynomial<aerobus::q32>↩
::template val< aerobus::make_q32_t<8, 375115>, aerobus::make_q32_t<30, 208117>, aerobus::make_q32_t<109,
81261>, aerobus::make_q32_t<5161841, 536870912>, aerobus::make_q32_t<3724869, 67108864>,
aerobus::make_q32_t<16121323, 67108864>, aerobus::make_q32_t<1453635, 2097152>, aerobus↩
::q32::one>
```

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

- src/aerobus.h

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

Used to evaluate polynomials over a value in Ring.

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

Classes

- struct inner
- struct inner< stop, stop >

8.23.1 Detailed Description

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

Used to evaluate polynomials over a value in Ring.

Template Parameters

P	a value in polynomial<Ring>

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

- src/aerobus.h

8.24 aerobus::i32 Struct Reference

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

`#include <aerobus.h>`

Classes

- struct val

  values in i32, again represented as types

Public Types

- using inner_type = int32_t
- using zero = val< 0 >

  constant zero
- using one = val< 1 >

  constant one
- template<auto x>
  using inject_constant_t = val< static_cast< int32_t >(x)>

  inject a native constant
- template<typename v >
  using inject_ring_t = v
- template<typename v1 , typename v2 >
  using add_t = typename add< v1, v2 >::type

  addition operator yields v1 + v2
- template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type

  substraction operator yields v1 - v2
- template<typename v1 , typename v2 >
  using mul_t = typename mul< v1, v2 >::type

  multiplication operator yields v1 ∗ v2
- template<typename v1 , typename v2 >
  using div_t = typename div< v1, v2 >::type

  division operator yields v1 / v2
- template<typename v1 , typename v2 >
  using mod_t = typename remainder< v1, v2 >::type

  modulus operator yields v1 % v2
- template<typename v1 , typename v2 >
  using gt_t = typename gt< v1, v2 >::type

  strictly greater operator (v1 > v2) yields v1 > v2
- template<typename v1 , typename v2 >
  using lt_t = typename lt< v1, v2 >::type

  strict less operator (v1 < v2) yields v1 < v2
- template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type

  equality operator (type) yields v1 == v2 as std::integral_constant<bool>
- template<typename v1 , typename v2 >
  using gcd_t = gcd_t< i32, v1, v2 >

  greatest common divisor yields GCD(v1, v2)
- template<typename v >
  using pos_t = typename pos< v >::type

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

Static Public Attributes

- static constexpr bool is_field = false

  integers are not a field
- static constexpr bool is_euclidean_domain = true

  integers are an euclidean domain
- template<typename v1 , typename v2 >
  static constexpr bool eq_v = eq_t<v1, v2>::value

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

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

8.24.1 Detailed Description

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

Examples

examples/compensated_horner.cpp.

8.24.2 Member Typedef Documentation

8.24.2.1 add_t

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

addition operator yields v1 + v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.2 div_t

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

division operator yields v1 / v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.3 eq_t

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

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

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.4 gcd_t

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

greatest common divisor yields GCD(v1, v2)

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.5 gt_t

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

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

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.6 inject_constant_t

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

inject a native constant

Template Parameters

x	

8.24.2.7 inject_ring_t

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

8.24.2.8 inner_type

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

8.24.2.9 lt_t

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

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

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.10 mod_t

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

modulus operator yields v1 % v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.11 mul_t

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

multiplication operator yields v1 $*$ v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.12 one

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

constant one

8.24.2.13 pos_t

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

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

Template Parameters

v	a value in i32

8.24.2.14 sub_t

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

substraction operator yields v1 - v2

Template Parameters

v1	a value in i32
v2	a value in i32

8.24.2.15 zero

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

constant zero

8.24.3 Member Data Documentation

8.24.3.1 eq_v

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

equality operator (boolean value)

Template Parameters

v1	
v2	

8.24.3.2 is_euclidean_domain

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

integers are an euclidean domain

8.24.3.3 is_field

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

integers are not a field

8.24.3.4 pos_v

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

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

Template Parameters

v	a value in i32

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

- src/aerobus.h

8.25 aerobus::i64 Struct Reference

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

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

Classes

- struct val
  values in i64

Public Types

- using inner_type = int64_t
  type of represented values
- template<auto x>
  using inject_constant_t = val< static_cast< int64_t >(x)>
  injects constant as an i64 value

- template<typename v >
  using inject_ring_t = v

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

  constant zero
- using one = val< 1 >

  constant one
- template<typename v1 , typename v2 >
  using add_t = typename add< v1, v2 >::type

  addition operator
- template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type

  substraction operator
- template<typename v1 , typename v2 >
  using mul_t = typename mul< v1, v2 >::type

  multiplication operator
- template<typename v1 , typename v2 >
  using div_t = typename div< v1, v2 >::type

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

  modulus operator
- template<typename v1 , typename v2 >
  using gt_t = typename gt< v1, v2 >::type

  strictly greater operator yields v1 > v2 as std::true_type or std::false_type
- template<typename v1 , typename v2 >
  using lt_t = typename lt< v1, v2 >::type

  strict less operator yields v1 < v2 as std::true_type or std::false_type
- template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type

  equality operator yields v1 == v2 as std::true_type or std::false_type
- template<typename v1 , typename v2 >
  using gcd_t = gcd_t< i64, v1, v2 >

  greatest common divisor yields GCD(v1, v2) as instanciation of i64::val
- template<typename v >
  using pos_t = typename pos< v >::type

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

Static Public Attributes

- static constexpr bool is_field = false

  integers are not a field
- static constexpr bool is_euclidean_domain = true

  integers are an euclidean domain
- template<typename v1 , typename v2 >
  static constexpr bool gt_v = gt_t<v1, v2>::value

  strictly greater operator yields v1 > v2 as boolean value
- template<typename v1 , typename v2 >
  static constexpr bool lt_v = lt_t<v1, v2>::value

  strictly smaller operator yields v1 < v2 as boolean value

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

  equality operator yields v1 == v2 as boolean value
- template<typename v >
  static constexpr bool pos_v = pos_t<v>::value

  positivity yields v > 0 as boolean value

8.25.1 Detailed Description

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

8.25.2 Member Typedef Documentation

8.25.2.1 add_t

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

addition operator

Template Parameters

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

8.25.2.2 div_t

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

division operator integer division

Template Parameters

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

8.25.2.3 eq_t

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

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

Template Parameters

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

8.25.2.4 gcd_t

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

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

Template Parameters

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

8.25.2.5 gt_t

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

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

Template Parameters

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

8.25.2.6 inject_constant_t

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

injects constant as an i64 value

Template Parameters

x	

8.25.2.7 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

8.25.2.8 inner_type

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

type of represented values

8.25.2.9 lt_t

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

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

Template Parameters

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

8.25.2.10 mod_t

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

modulus operator

Template Parameters

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

8.25.2.11 mul_t

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

multiplication operator

Template Parameters

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

8.25.2.12 one

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

constant one

8.25.2.13 pos_t

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

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

Template Parameters

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

8.25.2.14 sub_t

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

substraction operator

Template Parameters

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

8.25.2.15 zero

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

constant zero

8.25.3 Member Data Documentation

8.25.3.1 eq_v

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

equality operator yields v1 == v2 as boolean value

Template Parameters

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

8.25.3.2 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

8.25.3.3 is_euclidean_domain

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

integers are an euclidean domain

8.25.3.4 is_field

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

integers are not a field

8.25.3.5 lt_v

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

strictly smaller operator yields v1 $<$ v2 as boolean value

Template Parameters

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

8.25.3.6 pos_v

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

positivity yields v > 0 as boolean value

Template Parameters

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

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

- src/aerobus.h

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

`#include <aerobus.h>`

Public Types

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

8.26.1 Member Typedef Documentation

8.26.1.1 type

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

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

- src/aerobus.h

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

`#include <aerobus.h>`

Public Types

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

8.27.1 Member Typedef Documentation

8.27.1.1 type

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

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

- src/aerobus.h

8.28 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

8.28.1 Detailed Description

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

checks if n is prime

Template Parameters

n	

8.28.2 Member Data Documentation

8.28.2.1 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

8.29 aerobus::meta_libm< T > Struct Template Reference

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

Static Public Member Functions

- static INLINED DEVICE T floor (const T &f)
- static INLINED DEVICE T fmod (const T &x, const T &d)

8.29.1 Member Function Documentation

8.29.1.1 floor()

```
template<typename T >
static INLINED DEVICE T aerobus::meta_libm< T >::floor (
            const T & f )  [inline], [static]
```

8.29.1.2 fmod()

```
template<typename T >
static INLINED DEVICE T aerobus::meta_libm< T >::fmod (
            const T & x,
            const T & d )  [inline], [static]
```

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

- src/aerobus.h

8.30 aerobus::polynomial< Ring > Struct Template Reference

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

Classes

- struct horner_reduction_t

  Used to evaluate polynomials over a value in Ring.
- struct val

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

  specialization for constants

Public Types

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

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

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

  generator
- template<typename P >
  using simplify_t = typename simplify< P >::type

  simplifies a polynomial (recursively deletes highest degree if zero, do nothing otherwise)
- template<typename v1 , typename v2 >
  using add_t = typename add< v1, v2 >::type

  adds two polynomials
- template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type

  substraction of two polynomials
- template<typename v1 , typename v2 >
  using mul_t = typename mul< v1, v2 >::type

  multiplication of two polynomials
- template<typename v1 , typename v2 >
  using eq_t = typename eq_helper< v1, v2 >::type

  equality operator
- template<typename v1 , typename v2 >
  using 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 > >

  makes the constant (native type) polynomial a_0
- template<typename v >
  using inject_ring_t = val< v >

  makes the constant (ring type) polynomial a_0

Static Public Attributes

- static constexpr bool is_field = false
- static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain
- template<typename v >
  static constexpr bool pos_v = pos_t<v>::value

  positivity operator

8.30.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

Examples

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

8.30.2 Member Typedef Documentation

8.30.2.1 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	

8.30.2.2 derive_t

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

derivation operator

Template Parameters

v	

8.30.2.3 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	

8.30.2.4 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	

8.30.2.5 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	

8.30.2.6 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	

8.30.2.7 inject_constant_t

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

makes the constant (native type) polynomial a_0

Template Parameters

x	

8.30.2.8 inject_ring_t

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

makes the constant (ring type) polynomial a_0

Template Parameters

v	

8.30.2.9 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	

8.30.2.10 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	

8.30.2.11 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	

8.30.2.12 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	

8.30.2.13 one

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

constant one

8.30.2.14 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	

8.30.2.15 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	

8.30.2.16 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	

8.30.2.17 X

```
template<typename Ring >
```

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

generator

8.30.2.18 zero

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

constant zero

8.30.3 Member Data Documentation

8.30.3.1 is_euclidean_domain

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

8.30.3.2 is_field

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

8.30.3.3 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

8.31 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

8.31.1 Detailed Description

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

removes types from head of the list

8.31.2 Member Typedef Documentation

8.31.2.1 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

8.31.2.2 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

8.32 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 > >

  inject a 'constant' in quotient ring∗
- template<typename v >
  using inject_ring_t = val< v >

  projects a value of Ring onto the quotient

Static Public Attributes

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

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

  positivity operator always true
- static constexpr bool is_euclidean_domain = true

  quotien rings are euclidean domain

8.32.1 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>

8.32.2 Member Typedef Documentation

8.32.2.1 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

8.32.2.2 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

8.32.2.3 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

8.32.2.4 inject_constant_t

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

inject a 'constant' in quotient ring∗

Template Parameters

x	a 'constant' from Ring point of view

8.32.2.5 inject_ring_t

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

projects a value of Ring onto the quotient

Template Parameters

v	a value in Ring

8.32.2.6 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

8.32.2.7 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

8.32.2.8 one

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

one

8.32.2.9 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

8.32.2.10 zero

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

zero value

8.32.3 Member Data Documentation

8.32.3.1 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

8.32.3.2 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

8.32.3.3 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

8.33 aerobus::libm::internal::sin_poly< P > Struct Template Reference

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

- src/aerobus.h

8.34 aerobus::libm::internal::sin_poly< double > Struct Reference

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

Public Types

- using type = typename aerobus::polynomial< aerobus::q64 >::simplify_t< typename aerobus::polynomial< aerobus::q64 >::template val< aerobus::make_q64_t<-43, 1042171195712159 >, aerobus::make_q64_t<-89, 136637767615782 >, aerobus::make_q64_t< 123, 766493207966 >, aerobus::make_q64_t<-18133, 723813242548 >, aerobus::make_q64_t< 122341, 44395102410 >, aerobus::make_q64_t<-11252871, 56714469841 >, aerobus::make_q64_t< 2401919801264179, 288230376151711744 >, aerobus::make_q64_t<-6004799503160661, 36028797018963968 >, aerobus::q64::one > >

8.34.1 Member Typedef Documentation

8.34.1.1 type

```
using aerobus::libm::internal::sin_poly< double >::type = typename aerobus::polynomial<aerobus::q64>↩
::simplify_t< typename aerobus::polynomial<aerobus::q64>::  template val< aerobus::make_q64_t<-43,
1042171195712159>, aerobus::make_q64_t<-89, 136637767615782>, aerobus::make_q64_t<123, 766493207966>,
aerobus::make_q64_t<-18133, 723813242548>, aerobus::make_q64_t<122341, 44395102410>, aerobus::make_q64_t<-11
56714469841>, aerobus::make_q64_t<2401919801264179, 288230376151711744>, aerobus::make_q64_t<-60047995031606
36028797018963968>, aerobus::q64::one> >
```

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

- src/aerobus.h

8.35 aerobus::libm::internal::sin_poly< float > Struct Reference

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

Public Types

- using type = typename aerobus::polynomial< aerobus::q32 >::simplify_t< typename aerobus::polynomial< aerobus::q32 >::template val< aerobus::make_q32_t< 1, 357073 >, aerobus::make_q32_t<-67, 337533 >, aerobus::make_q32_t< 4473945, 536870912 >, aerobus::make_q32_t<-11184811, 67108864 >, aerobus::q32::one > >

8.35.1 Member Typedef Documentation

8.35.1.1 type

```
using aerobus::libm::internal::sin_poly< float >::type = typename aerobus::polynomial<aerobus::q32>↩
::simplify_t< typename aerobus::polynomial<aerobus::q32>::  template val< aerobus::make_q32_t<1,
357073>, aerobus::make_q32_t<-67, 337533>, aerobus::make_q32_t<4473945, 536870912>, aerobus::make_q32_t<-111
67108864>, aerobus::q32::one> >
```

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

- src/aerobus.h

8.36 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

8.36.1 Detailed Description

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

splits list at index

Template Parameters

index	

8.36.2 Member Typedef Documentation

8.36.2.1 head

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

8.36.2.2 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

8.37 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

8.37.1 Detailed Description

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

Empty pure template struct to handle type list.

A list of types.

Template Parameters

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

8.37.2 Member Typedef Documentation

8.37.2.1 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	

8.37.2.2 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	

8.37.2.3 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	

8.37.2.4 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	

8.37.2.5 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	

8.37.2.6 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	

8.37.3 Member Data Documentation

8.37.3.1 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

8.38 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

8.38.1 Detailed Description

specialization for empty type list

8.38.2 Member Typedef Documentation

8.38.2.1 concat

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

8.38.2.2 insert

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

8.38.2.3 push_back

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

8.38.2.4 push_front

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

8.38.3 Member Data Documentation

8.38.3.1 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

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

values in i32, again represented as types

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

Public Types

- using enclosing_type = i32
  Enclosing ring type.
- using is_zero_t = std::bool_constant< x==0 >
  is value zero

Static Public Member Functions

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

Static Public Attributes

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

8.39.1 Detailed Description

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

values in i32, again represented as types

Template Parameters

x	an actual integer

8.39.2 Member Typedef Documentation

8.39.2.1 enclosing_type

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

Enclosing ring type.

8.39.2.2 is_zero_t

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

is value zero

8.39.3 Member Function Documentation

8.39.3.1 get()

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

cast x into valueType

Template Parameters

valueType	double for example

8.39.3.2 to_string()

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

string representation of value

8.39.4 Member Data Documentation

8.39.4.1 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

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

values in i64

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

Public Types

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

Static Public Member Functions

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

Static Public Attributes

- static constexpr int64_t v = x
  actual value

8.40.1 Detailed Description

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

values in i64

Template Parameters

x	an actual integer

Examples

examples/compensated_horner.cpp.

8.40.2 Member Typedef Documentation

8.40.2.1 enclosing_type

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

enclosing ring type

8.40.2.2 inner_type

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

type of represented values

8.40.2.3 is_zero_t

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

is value zero

8.40.3 Member Function Documentation

8.40.3.1 get()

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

cast value in valueType

Template Parameters

valueType	(double for example)

8.40.3.2 to_string()

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

string representation

8.40.4 Member Data Documentation

8.40.4.1 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

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

values (seen as types) in polynomial ring

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

Public Types

- using ring_type = Ring
  ring coefficients live in
- using enclosing_type = polynomial< Ring >
  enclosing ring type
- using aN = coeffN
  heavy weight coefficient (non zero)
- using strip = val< coeffs... >
  remove largest coefficient
- using is_zero_t = std::bool_constant<(degree==0) &&(aN::is_zero_t::value)>
  true_type if polynomial is constant zero
- template<size_t index>
  using coeff_at_t = typename coeff_at< index >::type
  type of coefficient at index
- template<typename x >
  using value_at_t = horner_reduction_t< val > ::template inner< 0, degree+1 > ::template type< typename Ring::zero, x >

Static Public Member Functions

- static std::string to_string ()
  get a string representation of polynomial
- template<typename arithmeticType >
  static constexpr DEVICE INLINED arithmeticType eval (const arithmeticType &x)
  evaluates polynomial seen as a function operating on arithmeticType
- template<typename arithmeticType >
  static DEVICE INLINED arithmeticType compensated_eval (const arithmeticType &x)
  Evaluate polynomial on x using compensated horner scheme.

Static Public Attributes

- static constexpr size_t degree = sizeof...(coeffs)
  degree of the polynomial
- static constexpr bool is_zero_v = is_zero_t::value
  true if polynomial is constant zero

8.41.1 Detailed Description

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

values (seen as types) in polynomial ring

Template Parameters

coeffN	high degree coefficient
...coeffs	lower degree coefficients

Examples

examples/compensated_horner.cpp.

8.41.2 Member Typedef Documentation

8.41.2.1 aN

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

heavy weight coefficient (non zero)

8.41.2.2 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	

8.41.2.3 enclosing_type

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

enclosing ring type

8.41.2.4 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

8.41.2.5 ring_type

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

ring coefficients live in

8.41.2.6 strip

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

remove largest coefficient

8.41.2.7 value_at_t

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

8.41.3 Member Function Documentation

8.41.3.1 compensated_eval()

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

Evaluate polynomial on x using compensated horner scheme.

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

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

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

Template Parameters

arithmeticType	float for example

Parameters

x	

8.41.3.2 eval()

```
template<typename Ring >
template<typename coeffN , typename...  coeffs>
```

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

evaluates polynomial seen as a function operating on arithmeticType

Template Parameters

arithmeticType	usually float or double

Parameters

x	value

Returns

P(x)

8.41.3.3 to_string()

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

get a string representation of polynomial

Returns

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

8.41.4 Member Data Documentation

8.41.4.1 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

8.41.4.2 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

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

projection values in the quotient ring

`#include <aerobus.h>`

Public Types

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

8.42.1 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'

8.42.2 Member Typedef Documentation

8.42.2.1 raw_t

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

8.42.2.2 type

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

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

- src/aerobus.h

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

values in zpz

`#include <aerobus.h>`

Public Types

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

Static Public Member Functions

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

Static Public Attributes

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

8.43.1 Detailed Description

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

values in zpz

Template Parameters

x	an integer

8.43.2 Member Typedef Documentation

8.43.2.1 enclosing_type

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

enclosing ring type

8.43.2.2 is_zero_t

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

true_type if zero

8.43.3 Member Function Documentation

8.43.3.1 get()

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

get value as valueType

Template Parameters

valueType	an arithmetic type, such as float

8.43.3.2 to_string()

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

string representation

Returns

a string representation

8.43.4 Member Data Documentation

8.43.4.1 is_zero_v

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

true if zero

8.43.4.2 v

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

actual value

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

- src/aerobus.h

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

specialization for constants

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

Classes

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

Public Types

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

Static Public Member Functions

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

Static Public Attributes

- static constexpr size_t degree = 0

  degree
- static constexpr bool is_zero_v = is_zero_t::value

8.44.1 Detailed Description

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

specialization for constants

Template Parameters

coeffN	

8.44.2 Member Typedef Documentation

8.44.2.1 aN

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

8.44.2.2 coeff_at_t

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

8.44.2.3 enclosing_type

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

enclosing ring type

8.44.2.4 is_zero_t

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

8.44.2.5 ring_type

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

ring coefficients live in

8.44.2.6 strip

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

8.44.2.7 value_at_t

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

8.44.3 Member Function Documentation

8.44.3.1 compensated_eval()

```
template<typename Ring >
template<typename coeffN >
template<typename arithmeticType >
static DEVICE INLINED arithmeticType aerobus::polynomial< Ring >::val< coeffN >::compensated↩
_eval (
            const arithmeticType & x )  [inline], [static]
```

8.44.3.2 eval()

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

8.44.3.3 to_string()

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

8.44.4 Member Data Documentation

8.44.4.1 degree

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

degree

8.44.4.2 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

8.45 aerobus::zpz< p > Struct Template Reference

congruence classes of integers modulo p (32 bits)

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

Classes

- struct val
  values in zpz

Public Types

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

substraction operator

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

Static Public Attributes

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

8.45.1 Detailed Description

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

congruence classes of integers modulo p (32 bits)

if p is prime, zpz

is a field

Template Parameters

p	a integer

Examples

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

8.45.2 Member Typedef Documentation

8.45.2.1 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

8.45.2.2 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

8.45.2.3 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

8.45.2.4 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

8.45.2.5 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

8.45.2.6 inject_constant_t

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

injects a constant integer into zpz

Template Parameters

x	an integer

8.45.2.7 inner_type

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

underlying type for values

8.45.2.8 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

8.45.2.9 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

8.45.2.10 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

8.45.2.11 one

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

one value

8.45.2.12 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

8.45.2.13 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

8.45.2.14 zero

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

zero value

8.45.3 Member Data Documentation

8.45.3.1 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

8.45.3.2 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

8.45.3.3 is_euclidean_domain

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

always true

8.45.3.4 is_field

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

true iff p is prime

8.45.3.5 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

8.45.3.6 pos_v

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

positivity operator (boolean value)

Template Parameters

v1	a value in zpz::val

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

- src/aerobus.h

Chapter 9

File Documentation

9.1 README.md File Reference

9.2 src/aerobus.h File Reference

```
#include <cstdint>
#include <cstddef>
#include <cstring>
#include <type_traits>
#include <utility>
#include <algorithm>
#include <functional>
#include <string>
#include <concepts>
#include <array>
#include <cmath>
#include <limits>
```

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 #include <cmath>
00016 #include <limits>
00017 #ifdef WITH_CUDA_FP16
00018 #include <bit>
00019 #include <cuda_fp16.h>
00020 #endif
00021
00025 #ifdef _MSC_VER
```

```
00026 #define ALIGNED(x) __declspec(align(x))
00027 #define INLINED __forceinline
00028 #else
00029 #define ALIGNED(x) __attribute__((aligned(x)))
00030 #define INLINED __attribute__((always_inline)) inline
00031 #endif
00032
00033 #ifdef __CUDACC__
00034 #define DEVICE __host__ __device__
00035 #else
00036 #define DEVICE
00037 #endif
00038
00040
00042
00044
00045 // aligned allocation
00046 namespace aerobus {
00053     template<typename T>
00054     T* aligned_malloc(size_t count, size_t alignment) {
00055         #ifdef _MSC_VER
00056         return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00057         #else
00058         return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00059         #endif
00060     }
00061 }  // namespace aerobus
00062
00063 // concepts
00064 namespace aerobus {
00066     template <typename R>
00067     concept IsRing = requires {
00068         typename R::one;
00069         typename R::zero;
00070         typename R::template add_t<typename R::one, typename R::one>;
00071         typename R::template sub_t<typename R::one, typename R::one>;
00072         typename R::template mul_t<typename R::one, typename R::one>;
00073     };
00074
00076     template <typename R>
00077     concept IsEuclideanDomain = IsRing<R> && requires {
00078         typename R::template div_t<typename R::one, typename R::one>;
00079         typename R::template mod_t<typename R::one, typename R::one>;
00080         typename R::template gcd_t<typename R::one, typename R::one>;
00081         typename R::template eq_t<typename R::one, typename R::one>;
00082         typename R::template pos_t<typename R::one>;
00083
00084         R::template pos_v<typename R::one> == true;
00085         // typename R::template gt_t<typename R::one, typename R::zero>;
00086         R::is_euclidean_domain == true;
00087     };
00088
00090     template<typename R>
00091     concept IsField = IsEuclideanDomain<R> && requires {
00092         R::is_field == true;
00093     };
00094 }  // namespace aerobus
00095
00096 #ifdef WITH_CUDA_FP16
00097 // all this shit is required because of NVIDIA bug https://developer.nvidia.com/bugs/4863696
00098 namespace aerobus {
00099     namespace internal {
00100         static consteval DEVICE uint16_t my_internal_float2half(
00101             const float f, uint32_t &sign, uint32_t &remainder) {
00102             uint32_t x;
00103             uint32_t u;
00104             uint32_t result;
00105             x = std::bit_cast<int32_t>(f);
00106             u = (x & 0x7fffffffU);
00107             sign = ((x >> 16U) & 0x8000U);
00108             // NaN/+Inf/-Inf
00109             if (u >= 0x7f800000U) {
00110                 remainder = 0U;
00111                 result = ((u == 0x7f800000U) ? (sign | 0x7c00U) : 0x7fffU);
00112             } else if (u > 0x477fefffU) {  // Overflows
00113                 remainder = 0x80000000U;
00114                 result = (sign | 0x7bffU);
00115             } else if (u >= 0x38800000U) {  // Normal numbers
00116                 remainder = u << 19U;
00117                 u -= 0x38000000U;
00118                 result = (sign | (u >> 13U));
00119             } else if (u < 0x33000001U) {  // +0/-0
00120                 remainder = u;
00121                 result = sign;
00122             } else {  // Denormal numbers
00123                 const uint32_t exponent = u >> 23U;
00124                 const uint32_t shift = 0x7eU - exponent;
```

```
00125                 uint32_t mantissa = (u & 0x7fffffU);
00126                 mantissa |= 0x800000U;
00127                 remainder = mantissa << (32U - shift);
00128                 result = (sign | (mantissa >> shift));
00129                 result &= 0x0000FFFFU;
00130             }
00131             return static_cast<uint16_t>(result);
00132         }
00133 
00134         static consteval DEVICE __half my_float2half_rn(const float a) {
00135             __half val;
00136             __half_raw r;
00137             uint32_t sign = 0U;
00138             uint32_t remainder = 0U;
00139             r.x = my_internal_float2half(a, sign, remainder);
00140             if ((remainder > 0x80000000U) || ((remainder == 0x80000000U) && ((r.x & 0x1U) != 0U))) {
00141                 r.x++;
00142             }
00143 
00144             val = std::bit_cast<__half>(r);
00145             return val;
00146         }
00147 
00148         template <int16_t i>
00149         static constexpr __half convert_int16_to_half = my_float2half_rn(static_cast<float>(i));
00150 
00151 
00152         template <typename Out, int16_t x, typename E = void>
00153         struct int16_convert_helper;
00154 
00155         template <typename Out, int16_t x>
00156         struct int16_convert_helper<Out, x,
00157             std::enable_if_t<!std::is_same_v<Out, __half> && !std::is_same_v<Out, __half2>> {
00158             static constexpr Out value() {
00159                 return static_cast<Out>(x);
00160             }
00161         };
00162 
00163         template <int16_t x>
00164         struct int16_convert_helper<__half, x> {
00165             static constexpr __half value() {
00166                 return convert_int16_to_half<x>;
00167             }
00168         };
00169 
00170         template <int16_t x>
00171         struct int16_convert_helper<__half2, x> {
00172             static constexpr __half2 value() {
00173                 return __half2(convert_int16_to_half<x>, convert_int16_to_half<x>);
00174             }
00175         };
00176 
00177     }  // namespace internal
00178 }  // namespace aerobus
00179 #endif
00180 
00181 //  cast
00182 namespace aerobus {
00183     namespace internal {
00184         template<typename Out, typename In>
00185         struct staticcast {
00186             template<auto x>
00187             static consteval INLINED DEVICE Out func() {
00188                 return static_cast<Out>(x);
00189             }
00190         };
00191 
00192         #ifdef WITH_CUDA_FP16
00193         template<>
00194         struct staticcast<__half, int16_t> {
00195             template<int16_t x>
00196             static consteval INLINED DEVICE __half func() {
00197                 return int16_convert_helper<__half, x>::value();
00198             }
00199         };
00200 
00201         template<>
00202         struct staticcast<__half2, int16_t> {
00203             template<int16_t x>
00204             static consteval INLINED DEVICE __half2 func() {
00205                 return int16_convert_helper<__half2, x>::value();
00206             }
00207         };
00208         #endif
00209     }  // namespace internal
00210 }  // namespace aerobus
00211 
```

```
00212 // fma_helper, required because nvidia fails to reconstruct fma for fp16 types
00213 namespace aerobus {
00214     namespace internal {
00215         template<typename T>
00216         struct fma_helper;
00217 
00218         template<>
00219         struct fma_helper<double> {
00220             static constexpr INLINED DEVICE double eval(const double x, const double y, const double
      z) {
00221                 return x * y + z;
00222             }
00223         };
00224 
00225         template<>
00226         struct fma_helper<long double> {
00227             static constexpr INLINED DEVICE long double eval(
00228                 const long double x, const long double y, const long double z) {
00229                     return x * y + z;
00230             }
00231         };
00232 
00233         template<>
00234         struct fma_helper<float> {
00235             static constexpr INLINED DEVICE float eval(const float x, const float y, const float z) {
00236                 return x * y + z;
00237             }
00238         };
00239 
00240         template<>
00241         struct fma_helper<int32_t> {
00242             static constexpr INLINED DEVICE int16_t eval(const int16_t x, const int16_t y, const
      int16_t z) {
00243                 return x * y + z;
00244             }
00245         };
00246 
00247         template<>
00248         struct fma_helper<int16_t> {
00249             static constexpr INLINED DEVICE int32_t eval(const int32_t x, const int32_t y, const
      int32_t z) {
00250                 return x * y + z;
00251             }
00252         };
00253 
00254         template<>
00255         struct fma_helper<int64_t> {
00256             static constexpr INLINED DEVICE int64_t eval(const int64_t x, const int64_t y, const
      int64_t z) {
00257                 return x * y + z;
00258             }
00259         };
00260 
00261         #ifdef WITH_CUDA_FP16
00262         template<>
00263         struct fma_helper<__half> {
00264             static constexpr INLINED DEVICE __half eval(const __half x, const __half y, const __half
      z) {
00265                 #ifdef __CUDA_ARCH__
00266                 return __hfma(x, y, z);
00267                 #else
00268                 return x * y + z;
00269                 #endif
00270             }
00271         };
00272         template<>
00273         struct fma_helper<__half2> {
00274             static constexpr INLINED DEVICE __half2 eval(const __half2 x, const __half2 y, const
      __half2 z) {
00275                 #ifdef __CUDA_ARCH__
00276                 return __hfma2(x, y, z);
00277                 #else
00278                 return x * y + z;
00279                 #endif
00280             }
00281         };
00282         #endif
00283     }  // namespace internal
00284 }  // namespace aerobus
00285 
00286 namespace aerobus {
00287     namespace internal {
00288         struct double_double {};
00289 
00290         template<typename T>
00291         struct arithmetic_helpers;
00292 
```

```
00293         template<>
00294         struct arithmetic_helpers<double> {
00295             using integers = int64_t;
00296             using upper_type = double_double;
00297             static constexpr double one = 1.0;
00298             static constexpr double zero = 0.0;
00299             static constexpr double pi = 0x1.921fb54442d18p1;
00300             static constexpr double pi_2 = 0x1.921fb54442d18p0;
00301             static constexpr double pi_4 = 0x1.921fb54442d18p-1;
00302             static constexpr double two_pi = 0x1.921fb54442d18p2;
00303             static constexpr double inv_two_pi = 0x1.45f306dc9c883p-3;
00304             static constexpr double half = 0x1p-1;
00305         };
00306 
00307         template<>
00308         struct arithmetic_helpers<float> {
00309             using integers = int32_t;
00310             using upper_type = double;
00311             static constexpr double one = 1.0F;
00312             static constexpr double zero = 0.0F;
00313             static constexpr float pi = 0x1.921fb6p1f;
00314             static constexpr double pi_2 = 0x1.921fb6p0f;
00315             static constexpr double pi_4 = 0x1.921fb6p-1f;
00316             static constexpr double two_pi = 0x1.921fb6p2f;
00317             static constexpr double inv_two_pi = 0x1.45f306p-3f;
00318             static constexpr double half = 0x1p-1f;
00319         };
00320 
00321         #ifdef WITH_CUDA_FP16
00322         template<>
00323         struct arithmetic_helpers<__half> {
00324             using integers = int16_t;
00325             using upper_type = float;
00326             const __half one = CUDART_ONE_FP16;
00327             const __half zero = CUDART_ZERO_FP16;
00328             const __half pi = __half_raw(0x4248);
00329             const __half pi_2 =   __half_raw(0b0111111001001000);
00330             const __half pi_4 =   __half_raw(0b0111101001001000);
00331             const __half two_pi = __half_raw(0b0100111001001000);
00332             const __half inv_two_pi = __half_raw(0b0111000100011000);
00333             const __half half = __half_raw(0b0111100000000000);
00334         };
00335 
00336         template<>
00337         struct arithmetic_helpers<__half2> {
00338             using integers = int16_t;
00339             using upper_type = float;  // TODO(JeWaVe) : check for float2
00340             const __half2 one = __half2(CUDART_ONE_FP16, CUDART_ONE_FP16);
00341             const __half2 zero = __half2(CUDART_ZERO_FP16, CUDART_ZERO_FP16);
00342             const __half2 pi = __half2(__half_raw(0x4248), __half_raw(0x4248));
00343             const __half2 pi_2 = __half2(__half_raw(0b0111111001001000),
      __half_raw(0b0111111001001000));
00344             const __half2 pi_4 = __half2(__half_raw(0b0111101001001000),
      __half_raw(0b0111101001001000));
00345             const __half2 two_pi = __half2(__half_raw(0b0100111001001000),
      __half_raw(0b0100111001001000));
00346             const __half2 half = __half2(__half_raw(0b0111100000000000),
      __half_raw(0b0111100000000000));
00347         };
00348         #endif
00349     }  // namespace internal
00350 
00351     template<typename T>
00352     struct meta_libm {
00353         static INLINED DEVICE T floor(const T& f) {
00354             return std::floor(f);
00355         }
00356         static INLINED DEVICE T fmod(const T& x, const T& d) {
00357             return std::fmod(x, d);
00358         }
00359     };
00360 
00361     // TODO(JeWaVe) : investigate as hfloor is pure device -- should be replaced by something
      different and constexpr
00362     #ifdef WITH_CUDA_FP16
00363     template<>
00364     struct meta_libm<__half> {
00365         static INLINED __device__ __half floor(const __half& f) {
00366             return hfloor(f);
00367         }
00368 
00369         static INLINED __device__ __half fmod(const __half& x, const __half& d) {
00370             __half i = meta_libm<__half>::floor(x / d);
00371             return x - d * i;
00372         }
00373     };
00374 
```

```
00375     template<>
00376     struct meta_libm<__half2> {
00377         static INLINED __device__ __half2 floor(const __half2& f) {
00378             return h2floor(f);
00379         }
00380 
00381         static INLINED __device__ __half2 fmod(const __half2& x, const __half2& d) {
00382             __half2 i = meta_libm<__half2>::floor(x / d);
00383             return x - d * i;
00384         }
00385     };
00386     #endif
00387 }  // namespace aerobus
00388 
00389 // compensated horner utilities
00390 namespace aerobus {
00391     namespace internal {
00392         template <typename T>
00393         struct FloatLayout;
00394 
00395         #ifdef _MSC_VER
00396         template <>
00397         struct FloatLayout<long double> {
00398             static constexpr uint8_t exponent = 11;
00399             static constexpr uint8_t mantissa = 52;
00400             static constexpr uint8_t r = 27;  // ceil(mantissa/2)
00401             static constexpr long double shift = (1LL << r) + 1;
00402         };
00403         #else
00404         template <>
00405         struct FloatLayout<long double> {
00406             static constexpr uint8_t exponent = 15;
00407             static constexpr uint8_t mantissa = 64;
00408             static constexpr uint8_t r = 32;  // ceil(mantissa/2)
00409             static constexpr long double shift = (1LL << r) + 1;
00410         };
00411         #endif
00412 
00413         template <>
00414         struct FloatLayout<double> {
00415             static constexpr uint8_t exponent = 11;
00416             static constexpr uint8_t mantissa = 52;
00417             static constexpr uint8_t r = 27;  // ceil(mantissa/2)
00418             static constexpr double shift = (1LL << r) + 1;
00419         };
00420 
00421         template <>
00422         struct FloatLayout<float> {
00423             static constexpr uint8_t exponent = 8;
00424             static constexpr uint8_t mantissa = 23;
00425             static constexpr uint8_t r = 11;  // ceil(mantissa/2)
00426             static constexpr float shift = (1 << r) + 1;
00427         };
00428 
00429         #ifdef WITH_CUDA_FP16
00430         template<>
00431         struct FloatLayout<__half> {
00432             static constexpr uint8_t exponent = 5;
00433             static constexpr uint8_t mantissa = 10;
00434         };
00435 
00436         template<>
00437         struct FloatLayout<__half2> {
00438             static constexpr uint8_t exponent = 5;
00439             static constexpr uint8_t mantissa = 10;
00440         };
00441         #endif
00442 
00443         template<typename T>
00444         struct Split {
00445             static constexpr INLINED DEVICE void func(T a, T *x, T *y)  {
00446                 T z = a * FloatLayout<T>::shift;
00447                 *x = z - (z - a);
00448                 *y = a - *x;
00449             }
00450         };
00451 
00452         #ifdef WITH_CUDA_FP16
00453         template<>
00454         struct Split<__half> {
00455             static constexpr INLINED DEVICE void func(__half a, __half *x, __half *y) {
00456                 __half z = a * __half_raw(0x5280);  // TODO(JeWaVe): check this value
00457                 *x = z - (z - a);
00458                 *y = a - *x;
00459             }
00460         };
00461 
```

```
00462         template<>
00463         struct Split<__half2> {
00464             static constexpr INLINED DEVICE void func(__half2 a, __half2 *x, __half2 *y) {
00465                 __half2 z = a * __half2(__half_raw(0x5280), __half_raw(0x5280));  // TODO(JeWaVe):
    check this value
00466                 *x = z - (z - a);
00467                 *y = a - *x;
00468             }
00469         };
00470         #endif
00471
00472         template<typename T>
00473         static constexpr INLINED DEVICE void two_sum(T a, T b, T *x, T *y) {
00474             *x = a + b;
00475             T z = *x - a;
00476             *y = (a - (*x - z)) + (b - z);
00477         }
00478
00479         template<typename T>
00480         static constexpr INLINED DEVICE void two_prod(T a, T b, T *x, T *y) {
00481             *x = a * b;
00482             #ifdef __clang__
00483             *y = fma_helper<T>::eval(a, b, -*x);
00484             #else
00485             T ah, al, bh, bl;
00486             Split<T>::func(a, &ah, &al);
00487             Split<T>::func(b, &bh, &bl);
00488             *y = al * bl - (((*x - ah * bh) - al * bh) - ah * bl);
00489             #endif
00490         }
00491
00492         template<typename T, size_t N>
00493         static INLINED DEVICE T horner(T *p1, T *p2, T x) {
00494             T r = p1[0] + p2[0];
00495             for (int64_t i = N - 1; i >= 0; --i) {
00496                 r = r * x + p1[N - i] + p2[N - i];
00497             }
00498
00499             return r;
00500         }
00501     }  // namespace internal
00502 }  // namespace aerobus
00503
00504 // utilities
00505 namespace aerobus {
00506     namespace internal {
00507         template<template<typename...> typename TT, typename T>
00508         struct is_instantiation_of : std::false_type { };
00509
00510         template<template<typename...> typename TT, typename... Ts>
00511         struct is_instantiation_of<TT, TT<Ts...>> : std::true_type { };
00512
00513         template<template<typename...> typename TT, typename T>
00514         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00515
00516         template <int64_t i, typename T, typename... Ts>
00517         struct type_at {
00518             static_assert(i < sizeof...(Ts) + 1, "index out of range");
00519             using type = typename type_at<i - 1, Ts...>::type;
00520         };
00521
00522         template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00523             using type = T;
00524         };
00525
00526         template <size_t i, typename... Ts>
00527         using type_at_t = typename type_at<i, Ts...>::type;
00528
00529
00530         template<size_t n, size_t i, typename E = void>
00531         struct _is_prime {};
00532
00533         template<size_t i>
00534         struct _is_prime<0, i> {
00535             static constexpr bool value = false;
00536         };
00537
00538         template<size_t i>
00539         struct _is_prime<1, i> {
00540             static constexpr bool value = false;
00541         };
00542
00543         template<size_t i>
00544         struct _is_prime<2, i> {
00545             static constexpr bool value = true;
00546         };
00547
```

```
00548         template<size_t i>
00549         struct _is_prime<3, i> {
00550             static constexpr bool value = true;
00551         };
00552 
00553         template<size_t i>
00554         struct _is_prime<5, i> {
00555             static constexpr bool value = true;
00556         };
00557 
00558         template<size_t i>
00559         struct _is_prime<7, i> {
00560             static constexpr bool value = true;
00561         };
00562 
00563         template<size_t n, size_t i>
00564         struct _is_prime<n, i, std::enable_if_t<(n != 2 && n % 2 == 0)» {
00565             static constexpr bool value = false;
00566         };
00567 
00568         template<size_t n, size_t i>
00569         struct _is_prime<n, i, std::enable_if_t<(n != 2 && n != 3 && n % 2 != 0 && n % 3 == 0)» {
00570             static constexpr bool value = false;
00571         };
00572 
00573         template<size_t n, size_t i>
00574         struct _is_prime<n, i, std::enable_if_t<(n >= 9 && i * i > n)» {
00575             static constexpr bool value = true;
00576         };
00577 
00578         template<size_t n, size_t i>
00579         struct _is_prime<n, i, std::enable_if_t<(
00580             n % i == 0 &&
00581             n >= 9 &&
00582             n % 3 != 0 &&
00583             n % 2 != 0 &&
00584             i * i > n)» {
00585             static constexpr bool value = true;
00586         };
00587 
00588         template<size_t n, size_t i>
00589         struct _is_prime<n, i, std::enable_if_t<(
00590             n % (i+2) == 0 &&
00591             n >= 9 &&
00592             n % 3 != 0 &&
00593             n % 2 != 0 &&
00594             i * i <= n)» {
00595             static constexpr bool value = true;
00596         };
00597 
00598         template<size_t n, size_t i>
00599         struct _is_prime<n, i, std::enable_if_t<(
00600                 n % (i+2) != 0 &&
00601                 n % i != 0 &&
00602                 n >= 9 &&
00603                 n % 3 != 0 &&
00604                 n % 2 != 0 &&
00605                 (i * i <= n))» {
00606             static constexpr bool value = _is_prime<n, i+6>::value;
00607         };
00608     }  // namespace internal
00609 
00612     template<size_t n>
00613     struct is_prime {
00615         static constexpr bool value = internal::_is_prime<n, 5>::value;
00616     };
00617 
00621     template<size_t n>
00622     static constexpr bool is_prime_v = is_prime<n>::value;
00623 
00624     // gcd
00625     namespace internal {
00626         template <std::size_t... Is>
00627         constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
00628             -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
00629 
00630         template <std::size_t N>
00631         using make_index_sequence_reverse
00632             = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
00633 
00639         template<typename Ring, typename E = void>
00640         struct gcd;
00641 
00642         template<typename Ring>
00643         struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
00644             template<typename A, typename B, typename E = void>
00645             struct gcd_helper {};
```

```
00646
00647             // B = 0, A > 0
00648             template<typename A, typename B>
00649             struct gcd_helper<A, B, std::enable_if_t<
00650                 ((B::is_zero_t::value) &&
00651                     (Ring::template gt_t<A, typename Ring::zero>::value))» {
00652                 using type = A;
00653             };
00654
00655             // B = 0, A < 0
00656             template<typename A, typename B>
00657             struct gcd_helper<A, B, std::enable_if_t<
00658                 ((B::is_zero_t::value) &&
00659                     !(Ring::template gt_t<A, typename Ring::zero>::value))» {
00660                 using type = typename Ring::template sub_t<typename Ring::zero, A>;
00661             };
00662
00663             // B != 0
00664             template<typename A, typename B>
00665             struct gcd_helper<A, B, std::enable_if_t<
00666                 (!B::is_zero_t::value)
00667                 » {
00668             private: // NOLINT
00669                 // A / B
00670                 using k = typename Ring::template div_t<A, B>;
00671                 // A - (A/B)*B = A % B
00672                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
00673
00674             public:
00675                 using type = typename gcd_helper<B, m>::type;
00676             };
00677
00678             template<typename A, typename B>
00679             using type = typename gcd_helper<A, B>::type;
00680         };
00681     }  // namespace internal
00682
00683     // vadd and vmul
00684     namespace internal {
00685         template<typename... vals>
00686         struct vmul {};
00687
00688         template<typename v1, typename... vals>
00689         struct vmul<v1, vals...> {
00690             using type = typename v1::enclosing_type::template mul_t<v1, typename
      vmul<vals...>::type>;
00691         };
00692
00693         template<typename v1>
00694         struct vmul<v1> {
00695             using type = v1;
00696         };
00697
00698         template<typename... vals>
00699         struct vadd {};
00700
00701         template<typename v1, typename... vals>
00702         struct vadd<v1, vals...> {
00703             using type = typename v1::enclosing_type::template add_t<v1, typename
      vadd<vals...>::type>;
00704         };
00705
00706         template<typename v1>
00707         struct vadd<v1> {
00708             using type = v1;
00709         };
00710     }  // namespace internal
00711
00714     template<typename T, typename A, typename B>
00715     using gcd_t = typename internal::gcd<T>::template type<A, B>;
00716
00720     template<typename... vals>
00721     using vadd_t = typename internal::vadd<vals...>::type;
00722
00726     template<typename... vals>
00727     using vmul_t = typename internal::vmul<vals...>::type;
00728
00732     template<typename val>
00733     requires IsEuclideanDomain<typename val::enclosing_type>
00734     using abs_t = std::conditional_t<
00735                     val::enclosing_type::template pos_v<val>,
00736                     val, typename val::enclosing_type::template
      sub_t<typename val::enclosing_type::zero, val>>;
00737 }  // namespace aerobus
00738
00739 // embedding
00740 namespace aerobus {
```

```
00745     template<typename Small, typename Large, typename E = void>
00746     struct Embed;
00747 }  // namespace aerobus
00748
00749 namespace aerobus {
00754     template<typename Ring, typename X>
00755     requires IsRing<Ring>
00756     struct Quotient {
00759         template <typename V>
00760         struct val {
00761          public:
00762             using raw_t = V;
00763             using type = abs_t<typename Ring::template mod_t<V, X>>;
00764         };
00765
00767         using zero = val<typename Ring::zero>;
00768
00770         using one = val<typename Ring::one>;
00771
00775         template<typename v1, typename v2>
00776         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00777
00781         template<typename v1, typename v2>
00782         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00783
00787         template<typename v1, typename v2>
00788         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00789
00793         template<typename v1, typename v2>
00794         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00795
00799         template<typename v1, typename v2>
00800         using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00801
00805         template<typename v1, typename v2>
00806         static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00807
00811         template<typename v1>
00812         using pos_t = std::true_type;
00813
00817         template<typename v>
00818         static constexpr bool pos_v = pos_t<v>::value;
00819
00821         static constexpr bool is_euclidean_domain = true;
00822
00826         template<auto x>
00827         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00828
00832         template<typename v>
00833         using inject_ring_t = val<v>;
00834     };
00835
00839     template<typename Ring, typename X>
00840     struct Embed<Quotient<Ring, X>, Ring> {
00843         template<typename val>
00844         using type = typename val::raw_t;
00845     };
00846 }  // namespace aerobus
00847
00848 // type_list
00849 namespace aerobus {
00851     template <typename... Ts>
00852     struct type_list;
00853
00854     namespace internal {
00855         template <typename T, typename... Us>
00856         struct pop_front_h {
00857             using tail = type_list<Us...>;
00858             using head = T;
00859         };
00860
00861         template <size_t index, typename L1, typename L2>
00862         struct split_h {
00863          private:
00864             static_assert(index <= L2::length, "index ouf of bounds");
00865             using a = typename L2::pop_front::type;
00866             using b = typename L2::pop_front::tail;
00867             using c = typename L1::template push_back<a>;
00868
00869          public:
00870             using head = typename split_h<index - 1, c, b>::head;
00871             using tail = typename split_h<index - 1, c, b>::tail;
00872         };
00873
00874         template <typename L1, typename L2>
00875         struct split_h<0, L1, L2> {
00876             using head = L1;
```

```
00877             using tail = L2;
00878         };
00879
00880         template <size_t index, typename L, typename T>
00881         struct insert_h {
00882             static_assert(index <= L::length, "index ouf of bounds");
00883             using s = typename L::template split<index>;
00884             using left = typename s::head;
00885             using right = typename s::tail;
00886             using ll = typename left::template push_back<T>;
00887             using type = typename ll::template concat<right>;
00888         };
00889
00890         template <size_t index, typename L>
00891         struct remove_h {
00892             using s = typename L::template split<index>;
00893             using left = typename s::head;
00894             using right = typename s::tail;
00895             using rr = typename right::pop_front::tail;
00896             using type = typename left::template concat<rr>;
00897         };
00898     }  // namespace internal
00899
00902     template <typename... Ts>
00903     struct type_list {
00904      private:
00905         template <typename T>
00906         struct concat_h;
00907
00908         template <typename... Us>
00909         struct concat_h<type_list<Us...>> {
00910             using type = type_list<Ts..., Us...>;
00911         };
00912
00913      public:
00915         static constexpr size_t length = sizeof...(Ts);
00916
00919         template <typename T>
00920         using push_front = type_list<T, Ts...>;
00921
00924         template <size_t index>
00925         using at = internal::type_at_t<index, Ts...>;
00926
00928         struct pop_front {
00930             using type = typename internal::pop_front_h<Ts...>::head;
00932             using tail = typename internal::pop_front_h<Ts...>::tail;
00933         };
00934
00937         template <typename T>
00938         using push_back = type_list<Ts..., T>;
00939
00942         template <typename U>
00943         using concat = typename concat_h<U>::type;
00944
00947         template <size_t index>
00948         struct split {
00949          private:
00950             using inner = internal::split_h<index, type_list<>, type_list<Ts...>>;
00951
00952          public:
00953             using head = typename inner::head;
00954             using tail = typename inner::tail;
00955         };
00956
00960         template <typename T, size_t index>
00961         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
00962
00965         template <size_t index>
00966         using remove = typename internal::remove_h<index, type_list<Ts...>>::type;
00967     };
00968
00970     template <>
00971     struct type_list<> {
00972         static constexpr size_t length = 0;
00973
00974         template <typename T>
00975         using push_front = type_list<T>;
00976
00977         template <typename T>
00978         using push_back = type_list<T>;
00979
00980         template <typename U>
00981         using concat = U;
00982
00983         // TODO(jewave): assert index == 0
00984         template <typename T, size_t index>
00985         using insert = type_list<T>;
```

```
00986     };
00987 }  // namespace aerobus
00988
00989 // i16
00990 #ifdef WITH_CUDA_FP16
00991 // i16
00992 namespace aerobus {
00994     struct i16 {
00995         using inner_type = int16_t;
00998         template<int16_t x>
00999         struct val {
01001             using enclosing_type = i16;
01003             static constexpr int16_t v = x;
01004
01007             template<typename valueType>
01008             static constexpr INLINED DEVICE valueType get() {
01009                 return internal::template int16_convert_helper<valueType, x>::value();
01010             }
01011
01013             using is_zero_t = std::bool_constant<x == 0>;
01014
01016             static std::string to_string() {
01017                 return std::to_string(x);
01018             }
01019         };
01020
01022         using zero = val<0>;
01024         using one = val<1>;
01026         static constexpr bool is_field = false;
01028         static constexpr bool is_euclidean_domain = true;
01031         template<auto x>
01032         using inject_constant_t = val<static_cast<int16_t>(x)>;
01033
01034         template<typename v>
01035         using inject_ring_t = v;
01036
01037     private:
01038         template<typename v1, typename v2>
01039         struct add {
01040             using type = val<v1::v + v2::v>;
01041         };
01042
01043         template<typename v1, typename v2>
01044         struct sub {
01045             using type = val<v1::v - v2::v>;
01046         };
01047
01048         template<typename v1, typename v2>
01049         struct mul {
01050             using type = val<v1::v* v2::v>;
01051         };
01052
01053         template<typename v1, typename v2>
01054         struct div {
01055             using type = val<v1::v / v2::v>;
01056         };
01057
01058         template<typename v1, typename v2>
01059         struct remainder {
01060             using type = val<v1::v % v2::v>;
01061         };
01062
01063         template<typename v1, typename v2>
01064         struct gt {
01065             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01066         };
01067
01068         template<typename v1, typename v2>
01069         struct lt {
01070             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01071         };
01072
01073         template<typename v1, typename v2>
01074         struct eq {
01075             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01076         };
01077
01078         template<typename v1>
01079         struct pos {
01080             using type = std::bool_constant<(v1::v > 0)>;
01081         };
01082
01083     public:
01088         template<typename v1, typename v2>
01089         using add_t = typename add<v1, v2>::type;
01090
01095         template<typename v1, typename v2>
```

```
01096         using sub_t = typename sub<v1, v2>::type;
01097
01102         template<typename v1, typename v2>
01103         using mul_t = typename mul<v1, v2>::type;
01104
01109         template<typename v1, typename v2>
01110         using div_t = typename div<v1, v2>::type;
01111
01116         template<typename v1, typename v2>
01117         using mod_t = typename remainder<v1, v2>::type;
01118
01123         template<typename v1, typename v2>
01124         using gt_t = typename gt<v1, v2>::type;
01125
01130         template<typename v1, typename v2>
01131         using lt_t = typename lt<v1, v2>::type;
01132
01137         template<typename v1, typename v2>
01138         using eq_t = typename eq<v1, v2>::type;
01139
01143         template<typename v1, typename v2>
01144         static constexpr bool eq_v = eq_t<v1, v2>::value;
01145
01150         template<typename v1, typename v2>
01151         using gcd_t = gcd_t<i16, v1, v2>;
01152
01156         template<typename v>
01157         using pos_t = typename pos<v>::type;
01158
01162         template<typename v>
01163         static constexpr bool pos_v = pos_t<v>::value;
01164     };
01165 }  // namespace aerobus
01166 #endif
01167
01168 // i32
01169 namespace aerobus {
01171     struct i32 {
01172         using inner_type = int32_t;
01175         template<int32_t x>
01176         struct val {
01178             using enclosing_type = i32;
01180             static constexpr int32_t v = x;
01181
01184             template<typename valueType>
01185             static constexpr DEVICE valueType get() {
01186                 return static_cast<valueType>(x);
01187             }
01188
01190             using is_zero_t = std::bool_constant<x == 0>;
01191
01193             static std::string to_string() {
01194                 return std::to_string(x);
01195             }
01196         };
01197
01199         using zero = val<0>;
01201         using one = val<1>;
01203         static constexpr bool is_field = false;
01205         static constexpr bool is_euclidean_domain = true;
01208         template<auto x>
01209         using inject_constant_t = val<static_cast<int32_t>(x)>;
01210
01211         template<typename v>
01212         using inject_ring_t = v;
01213
01214      private:
01215         template<typename v1, typename v2>
01216         struct add {
01217             using type = val<v1::v + v2::v>;
01218         };
01219
01220         template<typename v1, typename v2>
01221         struct sub {
01222             using type = val<v1::v - v2::v>;
01223         };
01224
01225         template<typename v1, typename v2>
01226         struct mul {
01227             using type = val<v1::v* v2::v>;
01228         };
01229
01230         template<typename v1, typename v2>
01231         struct div {
01232             using type = val<v1::v / v2::v>;
01233         };
01234
```

```
01235         template<typename v1, typename v2>
01236         struct remainder {
01237             using type = val<v1::v % v2::v>;
01238         };
01239 
01240         template<typename v1, typename v2>
01241         struct gt {
01242             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01243         };
01244 
01245         template<typename v1, typename v2>
01246         struct lt {
01247             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01248         };
01249 
01250         template<typename v1, typename v2>
01251         struct eq {
01252             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01253         };
01254 
01255         template<typename v1>
01256         struct pos {
01257             using type = std::bool_constant<(v1::v > 0)>;
01258         };
01259 
01260     public:
01265         template<typename v1, typename v2>
01266         using add_t = typename add<v1, v2>::type;
01267 
01272         template<typename v1, typename v2>
01273         using sub_t = typename sub<v1, v2>::type;
01274 
01279         template<typename v1, typename v2>
01280         using mul_t = typename mul<v1, v2>::type;
01281 
01286         template<typename v1, typename v2>
01287         using div_t = typename div<v1, v2>::type;
01288 
01293         template<typename v1, typename v2>
01294         using mod_t = typename remainder<v1, v2>::type;
01295 
01300         template<typename v1, typename v2>
01301         using gt_t = typename gt<v1, v2>::type;
01302 
01307         template<typename v1, typename v2>
01308         using lt_t = typename lt<v1, v2>::type;
01309 
01314         template<typename v1, typename v2>
01315         using eq_t = typename eq<v1, v2>::type;
01316 
01320         template<typename v1, typename v2>
01321         static constexpr bool eq_v = eq_t<v1, v2>::value;
01322 
01327         template<typename v1, typename v2>
01328         using gcd_t = gcd_t<i32, v1, v2>;
01329 
01333         template<typename v>
01334         using pos_t = typename pos<v>::type;
01335 
01339         template<typename v>
01340         static constexpr bool pos_v = pos_t<v>::value;
01341     };
01342 }  // namespace aerobus
01343 
01344 // i64
01345 namespace aerobus {
01347     struct i64 {
01349         using inner_type = int64_t;
01352         template<int64_t x>
01353         struct val {
01355             using inner_type = int32_t;
01357             using enclosing_type = i64;
01359             static constexpr int64_t v = x;
01360 
01363             template<typename valueType>
01364             static constexpr INLINED DEVICE valueType get() {
01365                 return static_cast<valueType>(x);
01366             }
01367 
01369             using is_zero_t = std::bool_constant<x == 0>;
01370 
01372             static std::string to_string() {
01373                 return std::to_string(x);
01374             }
01375         };
01376 
01379         template<auto x>
```

```
01380         using inject_constant_t = val<static_cast<int64_t>(x)>;
01381
01386         template<typename v>
01387         using inject_ring_t = v;
01388
01390         using zero = val<0>;
01392         using one = val<1>;
01394         static constexpr bool is_field = false;
01396         static constexpr bool is_euclidean_domain = true;
01397
01398     private:
01399         template<typename v1, typename v2>
01400         struct add {
01401             using type = val<v1::v + v2::v>;
01402         };
01403
01404         template<typename v1, typename v2>
01405         struct sub {
01406             using type = val<v1::v - v2::v>;
01407         };
01408
01409         template<typename v1, typename v2>
01410         struct mul {
01411             using type = val<v1::v* v2::v>;
01412         };
01413
01414         template<typename v1, typename v2>
01415         struct div {
01416             using type = val<v1::v / v2::v>;
01417         };
01418
01419         template<typename v1, typename v2>
01420         struct remainder {
01421             using type = val<v1::v% v2::v>;
01422         };
01423
01424         template<typename v1, typename v2>
01425         struct gt {
01426             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
01427         };
01428
01429         template<typename v1, typename v2>
01430         struct lt {
01431             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
01432         };
01433
01434         template<typename v1, typename v2>
01435         struct eq {
01436             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
01437         };
01438
01439         template<typename v>
01440         struct pos {
01441             using type = std::bool_constant<(v::v > 0)>;
01442         };
01443
01444     public:
01448         template<typename v1, typename v2>
01449         using add_t = typename add<v1, v2>::type;
01450
01454         template<typename v1, typename v2>
01455         using sub_t = typename sub<v1, v2>::type;
01456
01460         template<typename v1, typename v2>
01461         using mul_t = typename mul<v1, v2>::type;
01462
01467         template<typename v1, typename v2>
01468         using div_t = typename div<v1, v2>::type;
01469
01473         template<typename v1, typename v2>
01474         using mod_t = typename remainder<v1, v2>::type;
01475
01480         template<typename v1, typename v2>
01481         using gt_t = typename gt<v1, v2>::type;
01482
01487         template<typename v1, typename v2>
01488         static constexpr bool gt_v = gt_t<v1, v2>::value;
01489
01494         template<typename v1, typename v2>
01495         using lt_t = typename lt<v1, v2>::type;
01496
01501         template<typename v1, typename v2>
01502         static constexpr bool lt_v = lt_t<v1, v2>::value;
01503
01508         template<typename v1, typename v2>
01509         using eq_t = typename eq<v1, v2>::type;
01510
```

```
01515         template<typename v1, typename v2>
01516         static constexpr bool eq_v = eq_t<v1, v2>::value;
01517
01522         template<typename v1, typename v2>
01523         using gcd_t = gcd_t<i64, v1, v2>;
01524
01528         template<typename v>
01529         using pos_t = typename pos<v>::type;
01530
01534         template<typename v>
01535         static constexpr bool pos_v = pos_t<v>::value;
01536     };
01537
01539     template<>
01540     struct Embed<i32, i64> {
01543         template<typename val>
01544         using type = i64::val<static_cast<int64_t>(val::v)>;
01545     };
01546 }  // namespace aerobus
01547
01548 // z/pz
01549 namespace aerobus {
01555     template<int32_t p>
01556     struct zpz {
01558         using inner_type = int32_t;
01559
01562         template<int32_t x>
01563         struct val {
01565             using enclosing_type = zpz<p>;
01567             static constexpr int32_t v = x % p;
01568
01571             template<typename valueType>
01572             static constexpr INLINED DEVICE valueType get() {
01573                 return static_cast<valueType>(x % p);
01574             }
01575
01577             using is_zero_t = std::bool_constant<v == 0>;
01578
01580             static constexpr bool is_zero_v = v == 0;
01581
01584             static std::string to_string() {
01585                 return std::to_string(x % p);
01586             }
01587         };
01588
01591         template<auto x>
01592         using inject_constant_t = val<static_cast<int32_t>(x)>;
01593
01595         using zero = val<0>;
01596
01598         using one = val<1>;
01599
01601         static constexpr bool is_field = is_prime<p>::value;
01602
01604         static constexpr bool is_euclidean_domain = true;
01605
01606      private:
01607         template<typename v1, typename v2>
01608         struct add {
01609             using type = val<(v1::v + v2::v) % p>;
01610         };
01611
01612         template<typename v1, typename v2>
01613         struct sub {
01614             using type = val<(v1::v - v2::v) % p>;
01615         };
01616
01617         template<typename v1, typename v2>
01618         struct mul {
01619             using type = val<(v1::v* v2::v) % p>;
01620         };
01621
01622         template<typename v1, typename v2>
01623         struct div {
01624             using type = val<(v1::v% p) / (v2::v % p)>;
01625         };
01626
01627         template<typename v1, typename v2>
01628         struct remainder {
01629             using type = val<(v1::v% v2::v) % p>;
01630         };
01631
01632         template<typename v1, typename v2>
01633         struct gt {
01634             using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01635         };
01636
```

```
01637         template<typename v1, typename v2>
01638         struct lt {
01639             using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01640         };
01641 
01642         template<typename v1, typename v2>
01643         struct eq {
01644             using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01645         };
01646 
01647         template<typename v1>
01648         struct pos {
01649             using type = std::bool_constant<(v1::v > 0)>;
01650         };
01651 
01652      public:
01656         template<typename v1, typename v2>
01657         using add_t = typename add<v1, v2>::type;
01658 
01662         template<typename v1, typename v2>
01663         using sub_t = typename sub<v1, v2>::type;
01664 
01668         template<typename v1, typename v2>
01669         using mul_t = typename mul<v1, v2>::type;
01670 
01674         template<typename v1, typename v2>
01675         using div_t = typename div<v1, v2>::type;
01676 
01680         template<typename v1, typename v2>
01681         using mod_t = typename remainder<v1, v2>::type;
01682 
01686         template<typename v1, typename v2>
01687         using gt_t = typename gt<v1, v2>::type;
01688 
01692         template<typename v1, typename v2>
01693         static constexpr bool gt_v = gt_t<v1, v2>::value;
01694 
01698         template<typename v1, typename v2>
01699         using lt_t = typename lt<v1, v2>::type;
01700 
01704         template<typename v1, typename v2>
01705         static constexpr bool lt_v = lt_t<v1, v2>::value;
01706 
01710         template<typename v1, typename v2>
01711         using eq_t = typename eq<v1, v2>::type;
01712 
01716         template<typename v1, typename v2>
01717         static constexpr bool eq_v = eq_t<v1, v2>::value;
01718 
01722         template<typename v1, typename v2>
01723         using gcd_t = gcd_t<i32, v1, v2>;
01724 
01727         template<typename v1>
01728         using pos_t = typename pos<v1>::type;
01729 
01732         template<typename v>
01733         static constexpr bool pos_v = pos_t<v>::value;
01734     };
01735 
01738     template<int32_t x>
01739     struct Embed<zpz<x>, i32> {
01742         template <typename val>
01743         using type = i32::val<val::v>;
01744     };
01745 }  // namespace aerobus
01746 
01747 // polynomial
01748 namespace aerobus {
01749     // coeffN x^N + ...
01754     template<typename Ring>
01755     requires IsEuclideanDomain<Ring>
01756     struct polynomial {
01757         static constexpr bool is_field = false;
01758         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01759 
01762         template<typename P>
01763         struct horner_reduction_t {
01764             template<size_t index, size_t stop>
01765             struct inner {
01766                 template<typename accum, typename x>
01767                 using type = typename horner_reduction_t<P>::template inner<index + 1, stop>
01768                     ::template type<
01769                         typename Ring::template add_t<
01770                             typename Ring::template mul_t<x, accum>,
01771                             typename P::template coeff_at_t<P::degree - index>
01772                         >, x>;
01773             };
```

```
01774
01775             template<size_t stop>
01776             struct inner<stop, stop> {
01777                 template<typename accum, typename x>
01778                 using type = accum;
01779             };
01780         };
01781
01785         template<typename coeffN, typename... coeffs>
01786         struct val {
01788             using ring_type = Ring;
01790             using enclosing_type = polynomial<Ring>;
01792             static constexpr size_t degree = sizeof...(coeffs);
01794             using aN = coeffN;
01796             using strip = val<coeffs...>;
01798             using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01800             static constexpr bool is_zero_v = is_zero_t::value;
01801
01802          private:
01803             template<size_t index, typename E = void>
01804             struct coeff_at {};
01805
01806             template<size_t index>
01807             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))>> {
01808                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01809             };
01810
01811             template<size_t index>
01812             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))>> {
01813                 using type = typename Ring::zero;
01814             };
01815
01816          public:
01819             template<size_t index>
01820             using coeff_at_t = typename coeff_at<index>::type;
01821
01824             static std::string to_string() {
01825                 return string_helper<coeffN, coeffs...>::func();
01826             }
01827
01832             template<typename arithmeticType>
01833             static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01834                 #ifdef WITH_CUDA_FP16
01835                 arithmeticType start;
01836                 if constexpr (std::is_same_v<arithmeticType, __half2>) {
01837                     start = __half2(0, 0);
01838                 } else {
01839                     start = static_cast<arithmeticType>(0);
01840                 }
01841                 #else
01842                 arithmeticType start = static_cast<arithmeticType>(0);
01843                 #endif
01844                 return horner_evaluation<arithmeticType, val>
01845                         ::template inner<0, degree + 1>
01846                         ::func(start, x);
01847             }
01848
01861             template<typename arithmeticType>
01862             static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01863                 return compensated_horner<arithmeticType, val>::func(x);
01864             }
01865
01866             template<typename x>
01867             using value_at_t = horner_reduction_t<val>
01868                 ::template inner<0, degree + 1>
01869                 ::template type<typename Ring::zero, x>;
01870         };
01871
01874         template<typename coeffN>
01875         struct val<coeffN> {
01877             using ring_type = Ring;
01879             using enclosing_type = polynomial<Ring>;
01881             static constexpr size_t degree = 0;
01882             using aN = coeffN;
01883             using strip = val<coeffN>;
01884             using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01885
01886             static constexpr bool is_zero_v = is_zero_t::value;
01887
01888             template<size_t index, typename E = void>
01889             struct coeff_at {};
01890
01891             template<size_t index>
01892             struct coeff_at<index, std::enable_if_t<(index == 0)>> {
01893                 using type = aN;
01894             };
01895
```

```
01896             template<size_t index>
01897             struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
01898                 using type = typename Ring::zero;
01899             };
01900 
01901             template<size_t index>
01902             using coeff_at_t = typename coeff_at<index>::type;
01903 
01904             static std::string to_string() {
01905                 return string_helper<coeffN>::func();
01906             }
01907 
01908             template<typename arithmeticType>
01909             static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& x) {
01910                 return coeffN::template get<arithmeticType>();
01911             }
01912 
01913             template<typename arithmeticType>
01914             static DEVICE INLINED arithmeticType compensated_eval(const arithmeticType& x) {
01915                 return coeffN::template get<arithmeticType>();
01916             }
01917 
01918             template<typename x>
01919             using value_at_t = coeffN;
01920         };
01921 
01923         using zero = val<typename Ring::zero>;
01925         using one = val<typename Ring::one>;
01927         using X = val<typename Ring::one, typename Ring::zero>;
01928 
01929     private:
01930         template<typename P, typename E = void>
01931         struct simplify;
01932 
01933         template <typename P1, typename P2, typename I>
01934         struct add_low;
01935 
01936         template<typename P1, typename P2>
01937         struct add {
01938             using type = typename simplify<typename add_low<
01939             P1,
01940             P2,
01941             internal::make_index_sequence_reverse<
01942             std::max(P1::degree, P2::degree) + 1
01943             »::type>::type;
01944         };
01945 
01946         template <typename P1, typename P2, typename I>
01947         struct sub_low;
01948 
01949         template <typename P1, typename P2, typename I>
01950         struct mul_low;
01951 
01952         template<typename v1, typename v2>
01953         struct mul {
01954                 using type = typename mul_low<
01955                     v1,
01956                     v2,
01957                     internal::make_index_sequence_reverse<
01958                     v1::degree + v2::degree + 1
01959                     »::type;
01960         };
01961 
01962         template<typename coeff, size_t deg>
01963         struct monomial;
01964 
01965         template<typename v, typename E = void>
01966         struct derive_helper {};
01967 
01968         template<typename v>
01969         struct derive_helper<v, std::enable_if_t<v::degree == 0» {
01970             using type = zero;
01971         };
01972 
01973         template<typename v>
01974         struct derive_helper<v, std::enable_if_t<v::degree != 0» {
01975             using type = typename add<
01976                 typename derive_helper<typename simplify<typename v::strip>::type>::type,
01977                 typename monomial<
01978                     typename Ring::template mul_t<
01979                         typename v::aN,
01980                         typename Ring::template inject_constant_t<(v::degree)>
01981                     >,
01982                     v::degree - 1
01983                 >::type
01984             >::type;
01985         };
```

```
01986
01987         template<typename v1, typename v2, typename E = void>
01988         struct eq_helper {};
01989
01990         template<typename v1, typename v2>
01991         struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01992             using type = std::false_type;
01993         };
01994
01995
01996         template<typename v1, typename v2>
01997         struct eq_helper<v1, v2, std::enable_if_t<
01998             v1::degree == v2::degree &&
01999             (v1::degree != 0 || v2::degree != 0) &&
02000             std::is_same<
02001             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
02002             std::false_type
02003             >::value
02004         >
02005         > {
02006             using type = std::false_type;
02007         };
02008
02009         template<typename v1, typename v2>
02010         struct eq_helper<v1, v2, std::enable_if_t<
02011             v1::degree == v2::degree &&
02012             (v1::degree != 0 || v2::degree != 0) &&
02013             std::is_same<
02014             typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
02015             std::true_type
02016             >::value
02017         » {
02018             using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
02019         };
02020
02021         template<typename v1, typename v2>
02022         struct eq_helper<v1, v2, std::enable_if_t<
02023             v1::degree == v2::degree &&
02024             (v1::degree == 0)
02025         » {
02026             using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
02027         };
02028
02029         template<typename v1, typename v2, typename E = void>
02030         struct lt_helper {};
02031
02032         template<typename v1, typename v2>
02033         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
02034             using type = std::true_type;
02035         };
02036
02037         template<typename v1, typename v2>
02038         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
02039             using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
02040         };
02041
02042         template<typename v1, typename v2>
02043         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
02044             using type = std::false_type;
02045         };
02046
02047         template<typename v1, typename v2, typename E = void>
02048         struct gt_helper {};
02049
02050         template<typename v1, typename v2>
02051         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
02052             using type = std::true_type;
02053         };
02054
02055         template<typename v1, typename v2>
02056         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
02057             using type = std::false_type;
02058         };
02059
02060         template<typename v1, typename v2>
02061         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
02062             using type = std::false_type;
02063         };
02064
02065         // when high power is zero : strip
02066         template<typename P>
02067         struct simplify<P, std::enable_if_t<
02068             std::is_same<
02069             typename Ring::zero,
02070             typename P::aN
02071             >::value && (P::degree > 0)
02072         » {
```

```
02073             using type = typename simplify<typename P::strip>::type;
02074         };
02075
02076         // otherwise : do nothing
02077         template<typename P>
02078         struct simplify<P, std::enable_if_t<
02079             !std::is_same<
02080             typename Ring::zero,
02081             typename P::aN
02082             >::value && (P::degree > 0)
02083         >> {
02084             using type = P;
02085         };
02086
02087         // do not simplify constants
02088         template<typename P>
02089         struct simplify<P, std::enable_if_t<P::degree == 0>> {
02090             using type = P;
02091         };
02092
02093         // addition at
02094         template<typename P1, typename P2, size_t index>
02095         struct add_at {
02096             using type =
02097                 typename Ring::template add_t<
02098                     typename P1::template coeff_at_t<index>,
02099                     typename P2::template coeff_at_t<index>>;
02100         };
02101
02102         template<typename P1, typename P2, size_t index>
02103         using add_at_t = typename add_at<P1, P2, index>::type;
02104
02105         template<typename P1, typename P2, std::size_t... I>
02106         struct add_low<P1, P2, std::index_sequence<I...>> {
02107             using type = val<add_at_t<P1, P2, I>...>;
02108         };
02109
02110         // substraction at
02111         template<typename P1, typename P2, size_t index>
02112         struct sub_at {
02113             using type =
02114                 typename Ring::template sub_t<
02115                     typename P1::template coeff_at_t<index>,
02116                     typename P2::template coeff_at_t<index>>;
02117         };
02118
02119         template<typename P1, typename P2, size_t index>
02120         using sub_at_t = typename sub_at<P1, P2, index>::type;
02121
02122         template<typename P1, typename P2, std::size_t... I>
02123         struct sub_low<P1, P2, std::index_sequence<I...>> {
02124             using type = val<sub_at_t<P1, P2, I>...>;
02125         };
02126
02127         template<typename P1, typename P2>
02128         struct sub {
02129             using type = typename simplify<typename sub_low<
02130             P1,
02131             P2,
02132             internal::make_index_sequence_reverse<
02133             std::max(P1::degree, P2::degree) + 1
02134             >::type>::type;
02135         };
02136
02137         // multiplication at
02138         template<typename v1, typename v2, size_t k, size_t index, size_t stop>
02139         struct mul_at_loop_helper {
02140             using type = typename Ring::template add_t<
02141                 typename Ring::template mul_t<
02142                 typename v1::template coeff_at_t<index>,
02143                 typename v2::template coeff_at_t<k - index>
02144                 >,
02145                 typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
02146             >;
02147         };
02148
02149         template<typename v1, typename v2, size_t k, size_t stop>
02150         struct mul_at_loop_helper<v1, v2, k, stop, stop> {
02151             using type = typename Ring::template mul_t<
02152                 typename v1::template coeff_at_t<stop>,
02153                 typename v2::template coeff_at_t<0>>;
02154         };
02155
02156         template <typename v1, typename v2, size_t k, typename E = void>
02157         struct mul_at {};
02158
02159         template<typename v1, typename v2, size_t k>
```

```
02160         struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)» {
02161             using type = typename Ring::zero;
02162         };
02163 
02164         template<typename v1, typename v2, size_t k>
02165         struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)» {
02166             using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
02167         };
02168 
02169         template<typename P1, typename P2, size_t index>
02170         using mul_at_t = typename mul_at<P1, P2, index>::type;
02171 
02172         template<typename P1, typename P2, std::size_t... I>
02173         struct mul_low<P1, P2, std::index_sequence<I...» {
02174             using type = val<mul_at_t<P1, P2, I>...>;
02175         };
02176 
02177         // division helper
02178         template< typename A, typename B, typename Q, typename R, typename E = void>
02179         struct div_helper {};
02180 
02181         template<typename A, typename B, typename Q, typename R>
02182         struct div_helper<A, B, Q, R, std::enable_if_t<
02183             (R::degree < B::degree) ||
02184             (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02185             using q_type = Q;
02186             using mod_type = R;
02187             using gcd_type = B;
02188         };
02189 
02190         template<typename A, typename B, typename Q, typename R>
02191         struct div_helper<A, B, Q, R, std::enable_if_t<
02192             (R::degree >= B::degree) &&
02193             !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
02194          private: // NOLINT
02195             using rN = typename R::aN;
02196             using bN = typename B::aN;
02197             using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
      B::degree>::type;
02198             using rr = typename sub<R, typename mul<pT, B>::type>::type;
02199             using qq = typename add<Q, pT>::type;
02200 
02201          public:
02202             using q_type = typename div_helper<A, B, qq, rr>::q_type;
02203             using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
02204             using gcd_type = rr;
02205         };
02206 
02207         template<typename A, typename B>
02208         struct div {
02209             static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
02210             using q_type = typename div_helper<A, B, zero, A>::q_type;
02211             using m_type = typename div_helper<A, B, zero, A>::mod_type;
02212         };
02213 
02214         template<typename P>
02215         struct make_unit {
02216             using type = typename div<P, val<typename P::aN»::q_type;
02217         };
02218 
02219         template<typename coeff, size_t deg>
02220         struct monomial {
02221             using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
02222         };
02223 
02224         template<typename coeff>
02225         struct monomial<coeff, 0> {
02226             using type = val<coeff>;
02227         };
02228 
02229         template<typename arithmeticType, typename P>
02230         struct horner_evaluation {
02231             template<size_t index, size_t stop>
02232             struct inner {
02233                 static constexpr DEVICE INLINED arithmeticType func(
02234                     const arithmeticType& accum, const arithmeticType& x) {
02235                     return horner_evaluation<arithmeticType, P>::template inner<index + 1,
      stop>::func(
02236                         internal::fma_helper<arithmeticType>::eval(
02237                             x,
02238                             accum,
02239                             P::template coeff_at_t<P::degree - index>::template
      get<arithmeticType>()), x);
02240                 }
02241             };
02242 
02243             template<size_t stop>
```

```
02244             struct inner<stop, stop> {
02245                 static constexpr DEVICE INLINED arithmeticType func(
02246                     const arithmeticType& accum, const arithmeticType& x) {
02247                     return accum;
02248                 }
02249             };
02250         };
02251 
02252         template<typename arithmeticType, typename P>
02253         struct compensated_horner {
02254             template<int64_t index, int ghost>
02255             struct EFTHorner {
02256                 static INLINED DEVICE void func(
02257                         arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
      *r) {
02258                     arithmeticType p;
02259                     internal::two_prod(*r, x, &p, pi + P::degree - index - 1);
02260                     constexpr arithmeticType coeff = P::template coeff_at_t<index>::template
      get<arithmeticType>();
02261                     internal::two_sum<arithmeticType>(
02262                         p, coeff,
02263                         r, sigma + P::degree - index - 1);
02264                     EFTHorner<index - 1, ghost>::func(x, pi, sigma, r);
02265                 }
02266             };
02267 
02268             template<int ghost>
02269             struct EFTHorner<-1, ghost> {
02270                 static INLINED DEVICE void func(
02271                         arithmeticType x, arithmeticType *pi, arithmeticType *sigma, arithmeticType
      *r) {
02272                 }
02273             };
02274 
02275             static INLINED DEVICE arithmeticType func(arithmeticType x) {
02276                 arithmeticType pi[P::degree], sigma[P::degree];
02277                 arithmeticType r = P::template coeff_at_t<P::degree>::template get<arithmeticType>();
02278                 EFTHorner<P::degree - 1, 0>::func(x, pi, sigma, &r);
02279                 arithmeticType c = internal::horner<arithmeticType, P::degree - 1>(pi, sigma, x);
02280                 return r + c;
02281             }
02282         };
02283 
02284         template<typename coeff, typename... coeffs>
02285         struct string_helper {
02286             static std::string func() {
02287                 std::string tail = string_helper<coeffs...>::func();
02288                 std::string result = "";
02289                 if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
02290                     return tail;
02291                 } else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
02292                     if (sizeof...(coeffs) == 1) {
02293                         result += "x";
02294                     } else {
02295                         result += "x^" + std::to_string(sizeof...(coeffs));
02296                     }
02297                 } else {
02298                     if (sizeof...(coeffs) == 1) {
02299                         result += coeff::to_string() + " x";
02300                     } else {
02301                         result += coeff::to_string()
02302                                 + " x^" + std::to_string(sizeof...(coeffs));
02303                     }
02304                 }
02305 
02306                 if (!tail.empty()) {
02307                     if (tail.at(0) != '-') {
02308                         result += " + " + tail;
02309                     } else {
02310                         result += " - " + tail.substr(1);
02311                     }
02312                 }
02313 
02314                 return result;
02315             }
02316         };
02317 
02318         template<typename coeff>
02319         struct string_helper<coeff> {
02320             static std::string func() {
02321                 if (!std::is_same<coeff, typename Ring::zero>::value) {
02322                     return coeff::to_string();
02323                 } else {
02324                     return "";
02325                 }
02326             }
02327         };
```

```
02328
02329     public:
02332         template<typename P>
02333         using simplify_t = typename simplify<P>::type;
02334
02338         template<typename v1, typename v2>
02339         using add_t = typename add<v1, v2>::type;
02340
02344         template<typename v1, typename v2>
02345         using sub_t = typename sub<v1, v2>::type;
02346
02350         template<typename v1, typename v2>
02351         using mul_t = typename mul<v1, v2>::type;
02352
02356         template<typename v1, typename v2>
02357         using eq_t = typename eq_helper<v1, v2>::type;
02358
02362         template<typename v1, typename v2>
02363         using lt_t = typename lt_helper<v1, v2>::type;
02364
02368         template<typename v1, typename v2>
02369         using gt_t = typename gt_helper<v1, v2>::type;
02370
02374         template<typename v1, typename v2>
02375         using div_t = typename div<v1, v2>::q_type;
02376
02380         template<typename v1, typename v2>
02381         using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
02382
02386         template<typename coeff, size_t deg>
02387         using monomial_t = typename monomial<coeff, deg>::type;
02388
02391         template<typename v>
02392         using derive_t = typename derive_helper<v>::type;
02393
02396         template<typename v>
02397         using pos_t = typename Ring::template pos_t<typename v::aN>;
02398
02401         template<typename v>
02402         static constexpr bool pos_v = pos_t<v>::value;
02403
02407         template<typename v1, typename v2>
02408         using gcd_t = std::conditional_t<
02409             Ring::is_euclidean_domain,
02410             typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
02411             void>;
02412
02415         template<auto x>
02416         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
02417
02420         template<typename v>
02421         using inject_ring_t = val<v>;
02422     };
02423 }  // namespace aerobus
02424
02425 // fraction field
02426 namespace aerobus {
02427     namespace internal {
02428         template<typename Ring, typename E = void>
02429         requires IsEuclideanDomain<Ring>
02430         struct _FractionField {};
02431
02432         template<typename Ring>
02433         requires IsEuclideanDomain<Ring>
02434         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
02436             static constexpr bool is_field = true;
02437             static constexpr bool is_euclidean_domain = true;
02438
02439          private:
02440             template<typename val1, typename val2, typename E = void>
02441             struct to_string_helper {};
02442
02443             template<typename val1, typename val2>
02444             struct to_string_helper <val1, val2,
02445                 std::enable_if_t<
02446                 Ring::template eq_t<
02447                 val2, typename Ring::one
02448                 >::value
02449                 >
02450             > {
02451                 static std::string func() {
02452                     return val1::to_string();
02453                 }
02454             };
02455
02456             template<typename val1, typename val2>
02457             struct to_string_helper<val1, val2,
```

```
02458                 std::enable_if_t<
02459                 !Ring::template eq_t<
02460                 val2,
02461                 typename Ring::one
02462                 >::value
02463                 >
02464             > {
02465                 static std::string func() {
02466                     return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
02467                 }
02468             };
02469 
02470         public:
02474             template<typename val1, typename val2>
02475             struct val {
02477                 using x = val1;
02479                 using y = val2;
02481                 using is_zero_t = typename val1::is_zero_t;
02483                 static constexpr bool is_zero_v = val1::is_zero_t::value;
02484 
02486                 using ring_type = Ring;
02487                 using enclosing_type = _FractionField<Ring>;
02488 
02491                 static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
02492 
02493                 template<typename valueType, int ghost = 0>
02494                 struct get_helper {
02495                     static constexpr INLINED DEVICE valueType get() {
02496                         return internal::staticcast<valueType, typename
    ring_type::inner_type>::template func<x::v>() /
02497                                internal::staticcast<valueType, typename ring_type::inner_type>::template
    func<y::v>();
02498                     }
02499                 };
02500 
02501                 #ifdef WITH_CUDA_FP16
02502                 template<int ghost>
02503                 struct get_helper<__half, ghost> {
02504                     static constexpr INLINED DEVICE __half get() {
02505                         return internal::my_float2half_rn(
02506                                internal::staticcast<float, typename ring_type::inner_type>::template
    func<x::v>() /
02507                                internal::staticcast<float, typename ring_type::inner_type>::template
    func<y::v>());
02508                     }
02509                 };
02510 
02511                 template<int ghost>
02512                 struct get_helper<__half2, ghost> {
02513                     static constexpr INLINED DEVICE __half2 get() {
02514                         constexpr __half tmp = internal::my_float2half_rn(
02515                                internal::staticcast<float, typename ring_type::inner_type>::template
    func<x::v>() /
02516                                internal::staticcast<float, typename ring_type::inner_type>::template
    func<y::v>());
02517                         return __half2(tmp, tmp);
02518                     }
02519                 };
02520                 #endif
02521 
02525                 template<typename valueType>
02526                 static constexpr INLINED DEVICE valueType get() {
02527                     return get_helper<valueType, 0>::get();
02528                 }
02529 
02532                 static std::string to_string() {
02533                     return to_string_helper<val1, val2>::func();
02534                 }
02535 
02540                 template<typename arithmeticType>
02541                 static constexpr DEVICE INLINED arithmeticType eval(const arithmeticType& v) {
02542                     return x::eval(v) / y::eval(v);
02543                 }
02544             };
02545 
02547             using zero = val<typename Ring::zero, typename Ring::one>;
02549             using one = val<typename Ring::one, typename Ring::one>;
02550 
02553             template<typename v>
02554             using inject_t = val<v, typename Ring::one>;
02555 
02558             template<auto x>
02559             using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
    Ring::one>;
02560 
02563             template<typename v>
02564             using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
```

```
02565
02567             using ring_type = Ring;
02568
02569          private:
02570             template<typename v, typename E = void>
02571             struct simplify {};
02572
02573             // x = 0
02574             template<typename v>
02575             struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {
02576                 using type = typename _FractionField<Ring>::zero;
02577             };
02578
02579             // x != 0
02580             template<typename v>
02581             struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {
02582              private:
02583                 using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
02584                 using newx = typename Ring::template div_t<typename v::x, _gcd>;
02585                 using newy = typename Ring::template div_t<typename v::y, _gcd>;
02586
02587                 using posx = std::conditional_t<
02588                                 !Ring::template pos_v<newy>,
02589                                 typename Ring::template sub_t<typename Ring::zero, newx>,
02590                                 newx>;
02591                 using posy = std::conditional_t<
02592                                 !Ring::template pos_v<newy>,
02593                                 typename Ring::template sub_t<typename Ring::zero, newy>,
02594                                 newy>;
02595              public:
02596                 using type = typename _FractionField<Ring>::template val<posx, posy>;
02597             };
02598
02599          public:
02602             template<typename v>
02603             using simplify_t = typename simplify<v>::type;
02604
02605          private:
02606             template<typename v1, typename v2>
02607             struct add {
02608              private:
02609                 using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02610                 using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02611                 using dividend = typename Ring::template add_t<a, b>;
02612                 using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02613                 using g = typename Ring::template gcd_t<dividend, diviser>;
02614
02615              public:
02616                 using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
      diviser»;
02617             };
02618
02619             template<typename v>
02620             struct pos {
02621                 using type = std::conditional_t<
02622                     (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
02623                     (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
02624                     std::true_type,
02625                     std::false_type>;
02626             };
02627
02628             template<typename v1, typename v2>
02629             struct sub {
02630              private:
02631                 using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02632                 using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02633                 using dividend = typename Ring::template sub_t<a, b>;
02634                 using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02635                 using g = typename Ring::template gcd_t<dividend, diviser>;
02636
02637              public:
02638                 using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
      diviser»;
02639             };
02640
02641             template<typename v1, typename v2>
02642             struct mul {
02643              private:
02644                 using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
02645                 using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
02646
02647              public:
02648                 using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
02649             };
02650
02651             template<typename v1, typename v2, typename E = void>
02652             struct div {};
```

```
02653
02654             template<typename v1, typename v2>
02655             struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
      _FractionField<Ring>::zero>::value>  {
02656              private:
02657                 using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
02658                 using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
02659
02660              public:
02661                 using type = typename _FractionField<Ring>::template simplify_t<val<a, b>>;
02662             };
02663
02664             template<typename v1, typename v2>
02665             struct div<v1, v2, std::enable_if_t<
02666                 std::is_same<zero, v1>::value && std::is_same<v2, zero>::value>  {
02667                 using type = one;
02668             };
02669
02670             template<typename v1, typename v2>
02671             struct eq {
02672                 using type = std::conditional_t<
02673                         std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
02674                         std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
02675                     std::true_type,
02676                     std::false_type>;
02677             };
02678
02679             template<typename v1, typename v2, typename E = void>
02680             struct gt;
02681
02682             template<typename v1, typename v2>
02683             struct gt<v1, v2, std::enable_if_t<
02684                 (eq<v1, v2>::type::value)
02685                 >> {
02686                 using type = std::false_type;
02687             };
02688
02689             template<typename v1, typename v2>
02690             struct gt<v1, v2, std::enable_if_t<
02691                 (!eq<v1, v2>::type::value) &&
02692                 (!pos<v1>::type::value) && (!pos<v2>::type::value)
02693                 >> {
02694                 using type = typename gt<
02695                     typename sub<zero, v1>::type, typename sub<zero, v2>::type
02696                 >::type;
02697             };
02698
02699             template<typename v1, typename v2>
02700             struct gt<v1, v2, std::enable_if_t<
02701                 (!eq<v1, v2>::type::value) &&
02702                 (pos<v1>::type::value) && (!pos<v2>::type::value)
02703                 >> {
02704                 using type = std::true_type;
02705             };
02706
02707             template<typename v1, typename v2>
02708             struct gt<v1, v2, std::enable_if_t<
02709                 (!eq<v1, v2>::type::value) &&
02710                 (!pos<v1>::type::value) && (pos<v2>::type::value)
02711                 >> {
02712                 using type = std::false_type;
02713             };
02714
02715             template<typename v1, typename v2>
02716             struct gt<v1, v2, std::enable_if_t<
02717                 (!eq<v1, v2>::type::value) &&
02718                 (pos<v1>::type::value) && (pos<v2>::type::value)
02719                 >> {
02720                 using type = typename Ring::template gt_t<
02721                     typename Ring::template mul_t<v1::x, v2::y>,
02722                     typename Ring::template mul_t<v2::y, v2::x>
02723                 >;
02724             };
02725
02726          public:
02730             template<typename v1, typename v2>
02731             using add_t = typename add<v1, v2>::type;
02732
02737             template<typename v1, typename v2>
02738             using mod_t = zero;
02739
02744             template<typename v1, typename v2>
02745             using gcd_t = v1;
02746
02750             template<typename v1, typename v2>
02751             using sub_t = typename sub<v1, v2>::type;
02752
```

```
02756             template<typename v1, typename v2>
02757             using mul_t = typename mul<v1, v2>::type;
02758
02762             template<typename v1, typename v2>
02763             using div_t = typename div<v1, v2>::type;
02764
02768             template<typename v1, typename v2>
02769             using eq_t = typename eq<v1, v2>::type;
02770
02774             template<typename v1, typename v2>
02775             static constexpr bool eq_v = eq<v1, v2>::type::value;
02776
02780             template<typename v1, typename v2>
02781             using gt_t = typename gt<v1, v2>::type;
02782
02786             template<typename v1, typename v2>
02787             static constexpr bool gt_v = gt<v1, v2>::type::value;
02788
02791             template<typename v1>
02792             using pos_t = typename pos<v1>::type;
02793
02796             template<typename v>
02797             static constexpr bool pos_v = pos_t<v>::value;
02798         };
02799
02800         template<typename Ring, typename E = void>
02801         requires IsEuclideanDomain<Ring>
02802         struct FractionFieldImpl {};
02803
02804         // fraction field of a field is the field itself
02805         template<typename Field>
02806         requires IsEuclideanDomain<Field>
02807         struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {
02808             using type = Field;
02809             template<typename v>
02810             using inject_t = v;
02811         };
02812
02813         // fraction field of a ring is the actual fraction field
02814         template<typename Ring>
02815         requires IsEuclideanDomain<Ring>
02816         struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
02817             using type = _FractionField<Ring>;
02818         };
02819     }  // namespace internal
02820
02823     template<typename Ring>
02824     requires IsEuclideanDomain<Ring>
02825     using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02826
02829     template<typename Ring>
02830     struct Embed<Ring, FractionField<Ring» {
02833         template<typename v>
02834         using type = typename FractionField<Ring>::template val<v, typename Ring::one>;
02835     };
02836 }  // namespace aerobus
02837
02838
02839 // short names for common types
02840 namespace aerobus {
02844     template<typename X, typename Y>
02845     requires IsRing<typename X::enclosing_type> &&
02846         (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02847     using add_t = typename X::enclosing_type::template add_t<X, Y>;
02848
02852     template<typename X, typename Y>
02853     requires IsRing<typename X::enclosing_type> &&
02854         (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02855     using sub_t = typename X::enclosing_type::template sub_t<X, Y>;
02856
02860     template<typename X, typename Y>
02861     requires IsRing<typename X::enclosing_type> &&
02862         (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02863     using mul_t = typename X::enclosing_type::template mul_t<X, Y>;
02864
02868     template<typename X, typename Y>
02869     requires IsEuclideanDomain<typename X::enclosing_type> &&
02870         (std::is_same_v<typename X::enclosing_type, typename Y::enclosing_type>)
02871     using div_t = typename X::enclosing_type::template div_t<X, Y>;
02872
02875     using q32 = FractionField<i32>;
02876
02879     using fpq32 = FractionField<polynomial<q32>>;
02880
02883     using q64 = FractionField<i64>;
02884
02886     using pi64 = polynomial<i64>;
```

```
02887
02889     using pq64 = polynomial<q64>;
02890
02892     using fpq64 = FractionField<polynomial<q64>>;
02893
02898     template<typename Ring, typename v1, typename v2>
02899     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02900
02907     template<typename v>
02908     using embed_int_poly_in_fractions_t =
02909             typename Embed<
02910                 polynomial<typename v::ring_type>,
02911                 polynomial<FractionField<typename v::ring_type>»::template type<v>;
02912
02916     template<int64_t p, int64_t q>
02917     using make_q64_t = typename q64::template simplify_t<
02918                 typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02919
02923     template<int32_t p, int32_t q>
02924     using make_q32_t = typename q32::template simplify_t<
02925                 typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02926
02927     #ifdef WITH_CUDA_FP16
02929     using q16 = FractionField<i16>;
02930
02934     template<int16_t p, int16_t q>
02935     using make_q16_t = typename q16::template simplify_t<
02936                 typename q16::val<i16::inject_constant_t<p>, i16::inject_constant_t<q»>;
02937
02938     #endif
02943     template<typename Ring, typename v1, typename v2>
02944     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02949     template<typename Ring, typename v1, typename v2>
02950     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02951
02953     template<>
02954     struct Embed<q32, q64> {
02957         template<typename v>
02958         using type = make_q64_t<static_cast<int64_t>(v::x::v), static_cast<int64_t>(v::y::v)>;
02959     };
02960
02964     template<typename Small, typename Large>
02965     struct Embed<polynomial<Small>, polynomial<Large» {
02966      private:
02967         template<typename v, typename i>
02968         struct at_low;
02969
02970         template<typename v, size_t i>
02971         struct at_index {
02972             using type = typename Embed<Small, Large>::template
      type<typename v::template coeff_at_t<i>>;
02973         };
02974
02975         template<typename v, size_t... Is>
02976         struct at_low<v, std::index_sequence<Is...» {
02977             using type = typename polynomial<Large>::template val<typename at_index<v, Is>::type...>;
02978         };
02979
02980      public:
02983         template<typename v>
02984         using type = typename at_low<v, typename internal::make_index_sequence_reverse<v::degree +
      1»::type;
02985     };
02986
02990     template<typename Ring, auto... xs>
02991     using make_int_polynomial_t = typename polynomial<Ring>::template val<
02992             typename Ring::template inject_constant_t<xs>...>;
02993
02997     template<typename Ring, auto... xs>
02998     using make_frac_polynomial_t = typename polynomial<FractionField<Ring>>::template val<
02999             typename FractionField<Ring>::template inject_constant_t<xs>...>;
03000 }  // namespace aerobus
03001
03002 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
03003 namespace aerobus {
03004     namespace internal {
03005         template<typename T, size_t x, typename E = void>
03006         struct factorial {};
03007
03008         template<typename T, size_t x>
03009         struct factorial<T, x, std::enable_if_t<(x > 0)» {
03010         private:
03011             template<typename, size_t, typename>
03012             friend struct factorial;
03013         public:
03014             using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
      x - 1>::type>;
```

```
03015             static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
03016         };
03017
03018         template<typename T>
03019         struct factorial<T, 0> {
03020          public:
03021             using type = typename T::one;
03022             static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
03023         };
03024     }  // namespace internal
03025
03029     template<typename T, size_t i>
03030     using factorial_t = typename internal::factorial<T, i>::type;
03031
03035     template<typename T, size_t i>
03036     inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
03037
03038     namespace internal {
03039         template<typename T, size_t k, size_t n, typename E = void>
03040         struct combination_helper {};
03041
03042         template<typename T, size_t k, size_t n>
03043         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
03044             using type = typename FractionField<T>::template mul_t<
03045                 typename combination_helper<T, k - 1, n - 1>::type,
03046                 makefraction_t<T, typename T::template val<n>, typename T::template val<k»>;
03047         };
03048
03049         template<typename T, size_t k, size_t n>
03050         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
03051             using type = typename combination_helper<T, n - k, n>::type;
03052         };
03053
03054         template<typename T, size_t n>
03055         struct combination_helper<T, 0, n> {
03056             using type = typename FractionField<T>::one;
03057         };
03058
03059         template<typename T, size_t k, size_t n>
03060         struct combination {
03061             using type = typename internal::combination_helper<T, k, n>::type::x;
03062             static constexpr typename T::inner_type value =
03063                         internal::combination_helper<T, k, n>::type::template get<typename
      T::inner_type>();
03064         };
03065     }  // namespace internal
03066
03069     template<typename T, size_t k, size_t n>
03070     using combination_t = typename internal::combination<T, k, n>::type;
03071
03076     template<typename T, size_t k, size_t n>
03077     inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
03078
03079     namespace internal {
03080         template<typename T, size_t m>
03081         struct bernoulli;
03082
03083         template<typename T, typename accum, size_t k, size_t m>
03084         struct bernoulli_helper {
03085             using type = typename bernoulli_helper<
03086                 T,
03087                 addfractions_t<T,
03088                     accum,
03089                     mulfractions_t<T,
03090                         makefraction_t<T,
03091                             combination_t<T, k, m + 1>,
03092                             typename T::one>,
03093                         typename bernoulli<T, k>::type
03094                     >
03095                 >,
03096                 k + 1,
03097                 m>::type;
03098         };
03099
03100         template<typename T, typename accum, size_t m>
03101         struct bernoulli_helper<T, accum, m, m> {
03102             using type = accum;
03103         };
03104
03105
03106
03107         template<typename T, size_t m>
03108         struct bernoulli {
03109             using type = typename FractionField<T>::template mul_t<
03110                 typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
```

```
03111                 makefraction_t<T,
03112                 typename T::template val<static_cast<typename T::inner_type>(-1)>,
03113                 typename T::template val<static_cast<typename T::inner_type>(m + 1)>
03114                 >
03115             >;
03116
03117             template<typename floatType>
03118             static constexpr floatType value = type::template get<floatType>();
03119         };
03120
03121         template<typename T>
03122         struct bernoulli<T, 0> {
03123             using type = typename FractionField<T>::one;
03124
03125             template<typename floatType>
03126             static constexpr floatType value = type::template get<floatType>();
03127         };
03128     }  // namespace internal
03129
03133     template<typename T, size_t n>
03134     using bernoulli_t = typename internal::bernoulli<T, n>::type;
03135
03140     template<typename FloatType, typename T, size_t n >
03141     inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
03142
03143     // bell numbers
03144     namespace internal {
03145         template<typename T, size_t n, typename E = void>
03146         struct bell_helper;
03147
03148         template <typename T, size_t n>
03149         struct bell_helper<T, n, std::enable_if_t<(n > 1)» {
03150             template<typename accum, size_t i, size_t stop>
03151             struct sum_helper {
03152              private:
03153                 using left = typename T::template mul_t<
03154                             combination_t<T, i, n-1>,
03155                             typename bell_helper<T, i>::type>;
03156                 using new_accum = typename T::template add_t<accum, left>;
03157              public:
03158                 using type = typename sum_helper<new_accum, i+1, stop>::type;
03159             };
03160
03161             template<typename accum, size_t stop>
03162             struct sum_helper<accum, stop, stop> {
03163                 using type = accum;
03164             };
03165
03166             using type = typename sum_helper<typename T::zero, 0, n>::type;
03167         };
03168
03169         template<typename T>
03170         struct bell_helper<T, 0> {
03171             using type = typename T::one;
03172         };
03173
03174         template<typename T>
03175         struct bell_helper<T, 1> {
03176             using type = typename T::one;
03177         };
03178     }  // namespace internal
03179
03183     template<typename T, size_t n>
03184     using bell_t = typename internal::bell_helper<T, n>::type;
03185
03189     template<typename T, size_t n>
03190     static constexpr typename T::inner_type bell_v = bell_t<T, n>::v;
03191
03192     namespace internal {
03193         template<typename T, int k, typename E = void>
03194         struct alternate {};
03195
03196         template<typename T, int k>
03197         struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
03198             using type = typename T::one;
03199             static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
03200         };
03201
03202         template<typename T, int k>
03203         struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
03204             using type = typename T::template sub_t<typename T::zero, typename T::one>;
03205             static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
03206         };
03207     }  // namespace internal
03208
```

```
03211     template<typename T, int k>
03212     using alternate_t = typename internal::alternate<T, k>::type;
03213
03216     template<typename T, size_t k>
03217     inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
03218
03219     namespace internal {
03220         template<typename T, int n, int k, typename E = void>
03221         struct stirling_1_helper {};
03222
03223         template<typename T>
03224         struct stirling_1_helper<T, 0, 0> {
03225             using type = typename T::one;
03226         };
03227
03228         template<typename T, int n>
03229         struct stirling_1_helper<T, n, 0, std::enable_if_t<(n > 0)>> {
03230             using type = typename T::zero;
03231         };
03232
03233         template<typename T, int n>
03234         struct stirling_1_helper<T, 0, n, std::enable_if_t<(n > 0)>> {
03235             using type = typename T::zero;
03236         };
03237
03238         template<typename T, int n, int k>
03239         struct stirling_1_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)>> {
03240             using type = typename T::template sub_t<
03241                             typename stirling_1_helper<T, n-1, k-1>::type,
03242                             typename T::template mul_t<
03243                                 typename T::template inject_constant_t<n-1>,
03244                                 typename stirling_1_helper<T, n-1, k>::type
03245                             >;
03246         };
03247     }  // namespace internal
03248
03253     template<typename T, int n, int k>
03254     using stirling_1_signed_t = typename internal::stirling_1_helper<T, n, k>::type;
03255
03260     template<typename T, int n, int k>
03261     using stirling_1_unsigned_t = abs_t<typename internal::stirling_1_helper<T, n, k>::type>;
03262
03267     template<typename T, int n, int k>
03268     static constexpr typename T::inner_type stirling_1_unsigned_v = stirling_1_unsigned_t<T, n, k>::v;
03269
03274     template<typename T, int n, int k>
03275     static constexpr typename T::inner_type stirling_1_signed_v = stirling_1_signed_t<T, n, k>::v;
03276
03277     namespace internal {
03278         template<typename T, int n, int k, typename E = void>
03279         struct stirling_2_helper {};
03280
03281         template<typename T, int n>
03282         struct stirling_2_helper<T, n, n, std::enable_if_t<(n >= 0)>> {
03283             using type = typename T::one;
03284         };
03285
03286         template<typename T, int n>
03287         struct stirling_2_helper<T, n, 0, std::enable_if_t<(n > 0)>> {
03288             using type = typename T::zero;
03289         };
03290
03291         template<typename T, int n>
03292         struct stirling_2_helper<T, 0, n, std::enable_if_t<(n > 0)>> {
03293             using type = typename T::zero;
03294         };
03295
03296         template<typename T, int n, int k>
03297         struct stirling_2_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0) && (k < n)>> {
03298             using type = typename T::template add_t<
03299                             typename stirling_2_helper<T, n-1, k-1>::type,
03300                             typename T::template mul_t<
03301                                 typename T::template inject_constant_t<k>,
03302                                 typename stirling_2_helper<T, n-1, k>::type
03303                             >;
03304         };
03305     }  // namespace internal
03306
03311     template<typename T, int n, int k>
03312     using stirling_2_t = typename internal::stirling_2_helper<T, n, k>::type;
03313
03318     template<typename T, int n, int k>
03319     static constexpr typename T::inner_type stirling_2_v = stirling_2_t<T, n, k>::v;
03320
03321     namespace internal {
03322         template<typename T>
03323         struct pow_scalar {
```

```
03324             template<size_t p>
03325             static constexpr DEVICE INLINED T func(const T& x) { return p == 0 ? static_cast<T>(1) :
03326                 p % 2 == 0 ? func<p/2>(x) * func<p/2>(x) :
03327                 x * func<p/2>(x) * func<p/2>(x);
03328             }
03329         };
03330 
03331         template<typename T, typename p, size_t n, typename E = void>
03332         requires IsEuclideanDomain<T>
03333         struct pow;
03334 
03335         template<typename T, typename p, size_t n>
03336         struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)>> {
03337             using type = typename T::template mul_t<
03338                 typename pow<T, p, n/2>::type,
03339                 typename pow<T, p, n/2>::type
03340             >;
03341         };
03342 
03343         template<typename T, typename p, size_t n>
03344         struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)>> {
03345             using type = typename T::template mul_t<
03346                 p,
03347                 typename T::template mul_t<
03348                     typename pow<T, p, n/2>::type,
03349                     typename pow<T, p, n/2>::type
03350                 >
03351             >;
03352         };
03353 
03354         template<typename T, typename p, size_t n>
03355         struct pow<T, p, n, std::enable_if_t<n == 0>> { using type = typename T::one; };
03356     }  // namespace internal
03357 
03362     template<typename T, typename p, size_t n>
03363     using pow_t = typename internal::pow<T, p, n>::type;
03364 
03369     template<typename T, typename p, size_t n>
03370     static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
03371 
03372     template<typename T, size_t p>
03373     static constexpr DEVICE INLINED T pow_scalar(const T& x) { return
      internal::pow_scalar<T>::template func<p>(x); }
03374 
03375     namespace internal {
03376         template<typename, template<typename, size_t> typename, class>
03377         struct make_taylor_impl;
03378 
03379         template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
03380         struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...>> {
03381             using type = typename polynomial<FractionField<T>>::template val<typename coeff_at<T,
      Is>::type...>;
03382         };
03383     }
03384 
03389     template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
03390     using taylor = typename internal::make_taylor_impl<
03391         T,
03392         coeff_at,
03393         internal::make_index_sequence_reverse<deg + 1>>::type;
03394 
03395     namespace internal {
03396         template<typename T, size_t i>
03397         struct exp_coeff {
03398             using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03399         };
03400 
03401         template<typename T, size_t i, typename E = void>
03402         struct sin_coeff_helper {};
03403 
03404         template<typename T, size_t i>
03405         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03406             using type = typename FractionField<T>::zero;
03407         };
03408 
03409         template<typename T, size_t i>
03410         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
03411             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03412         };
03413 
03414         template<typename T, size_t i>
03415         struct sin_coeff {
03416             using type = typename sin_coeff_helper<T, i>::type;
03417         };
03418 
03419         template<typename T, size_t i, typename E = void>
03420         struct sh_coeff_helper {};
```

```
03421
03422         template<typename T, size_t i>
03423         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03424             using type = typename FractionField<T>::zero;
03425         };
03426
03427         template<typename T, size_t i>
03428         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03429             using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03430         };
03431
03432         template<typename T, size_t i>
03433         struct sh_coeff {
03434             using type = typename sh_coeff_helper<T, i>::type;
03435         };
03436
03437         template<typename T, size_t i, typename E = void>
03438         struct cos_coeff_helper {};
03439
03440         template<typename T, size_t i>
03441         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03442             using type = typename FractionField<T>::zero;
03443         };
03444
03445         template<typename T, size_t i>
03446         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03447             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
03448         };
03449
03450         template<typename T, size_t i>
03451         struct cos_coeff {
03452             using type = typename cos_coeff_helper<T, i>::type;
03453         };
03454
03455         template<typename T, size_t i, typename E = void>
03456         struct cosh_coeff_helper {};
03457
03458         template<typename T, size_t i>
03459         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03460             using type = typename FractionField<T>::zero;
03461         };
03462
03463         template<typename T, size_t i>
03464         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03465             using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
03466         };
03467
03468         template<typename T, size_t i>
03469         struct cosh_coeff {
03470             using type = typename cosh_coeff_helper<T, i>::type;
03471         };
03472
03473         template<typename T, size_t i>
03474         struct geom_coeff { using type = typename FractionField<T>::one; };
03475
03476
03477         template<typename T, size_t i, typename E = void>
03478         struct atan_coeff_helper;
03479
03480         template<typename T, size_t i>
03481         struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03482             using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i»;
03483         };
03484
03485         template<typename T, size_t i>
03486         struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
03487             using type = typename FractionField<T>::zero;
03488         };
03489
03490         template<typename T, size_t i>
03491         struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
03492
03493         template<typename T, size_t i, typename E = void>
03494         struct asin_coeff_helper;
03495
03496         template<typename T, size_t i>
03497         struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
03498             using type = makefraction_t<T,
03499                 factorial_t<T, i - 1>,
03500                 typename T::template mul_t<
03501                     typename T::template val<i>,
03502                     T::template mul_t<
03503                         pow_t<T, typename T::template inject_constant_t<4>, i / 2>,
03504                         pow<T, factorial_t<T, i / 2>, 2
03505                     >
03506                 >
03507                 »;
```

```
03508         };
03509
03510         template<typename T, size_t i>
03511         struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03512             using type = typename FractionField<T>::zero;
03513         };
03514
03515         template<typename T, size_t i>
03516         struct asin_coeff {
03517             using type = typename asin_coeff_helper<T, i>::type;
03518         };
03519
03520         template<typename T, size_t i>
03521         struct lnp1_coeff {
03522             using type = makefraction_t<T,
03523                 alternate_t<T, i + 1>,
03524                 typename T::template val<i>;
03525         };
03526
03527         template<typename T>
03528         struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
03529
03530         template<typename T, size_t i, typename E = void>
03531         struct asinh_coeff_helper;
03532
03533         template<typename T, size_t i>
03534         struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
03535             using type = makefraction_t<T,
03536                 typename T::template mul_t<
03537                     alternate_t<T, i / 2>,
03538                     factorial_t<T, i - 1>
03539                 >,
03540                 typename T::template mul_t<
03541                     typename T::template mul_t<
03542                         typename T::template val<i>,
03543                         pow_t<T, factorial_t<T, i / 2>, 2>
03544                     >,
03545                     pow_t<T, typename T::template inject_constant_t<4>, i / 2>
03546                 >
03547             >;
03548         };
03549
03550         template<typename T, size_t i>
03551         struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03552             using type = typename FractionField<T>::zero;
03553         };
03554
03555         template<typename T, size_t i>
03556         struct asinh_coeff {
03557             using type = typename asinh_coeff_helper<T, i>::type;
03558         };
03559
03560         template<typename T, size_t i, typename E = void>
03561         struct atanh_coeff_helper;
03562
03563         template<typename T, size_t i>
03564         struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
03565             // 1/i
03566             using type = typename FractionField<T>:: template val<
03567                 typename T::one,
03568                 typename T::template inject_constant_t<i>;
03569         };
03570
03571         template<typename T, size_t i>
03572         struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
03573             using type = typename FractionField<T>::zero;
03574         };
03575
03576         template<typename T, size_t i>
03577         struct atanh_coeff {
03578             using type = typename atanh_coeff_helper<T, i>::type;
03579         };
03580
03581         template<typename T, size_t i, typename E = void>
03582         struct tan_coeff_helper;
03583
03584         template<typename T, size_t i>
03585         struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0>> {
03586             using type = typename FractionField<T>::zero;
03587         };
03588
03589         template<typename T, size_t i>
03590         struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0>> {
03591         private:
03592             // 4^((i+1)/2)
03593             using _4p = typename FractionField<T>::template inject_t<
03594                 pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2>;
```

```
03595             // 4^((i+1)/2) - 1
03596             using _4pm1 = typename FractionField<T>::template
      sub_t<_4p, typename FractionField<T>::one>;
03597             // (-1)^((i-1)/2)
03598             using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
03599             using dividend = typename FractionField<T>::template mul_t<
03600                 altp,
03601                 FractionField<T>::template mul_t<
03602                 _4p,
03603                 FractionField<T>::template mul_t<
03604                 _4pm1,
03605                 bernoulli_t<T, (i + 1)>
03606                 >
03607                 >
03608             >;
03609         public:
03610             using type = typename FractionField<T>::template div_t<dividend,
03611                 typename FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
03612         };
03613
03614         template<typename T, size_t i>
03615         struct tan_coeff {
03616             using type = typename tan_coeff_helper<T, i>::type;
03617         };
03618
03619         template<typename T, size_t i, typename E = void>
03620         struct tanh_coeff_helper;
03621
03622         template<typename T, size_t i>
03623         struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
03624             using type = typename FractionField<T>::zero;
03625         };
03626
03627         template<typename T, size_t i>
03628         struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
03629         private:
03630             using _4p = typename FractionField<T>::template inject_t<
03631                 pow_t<T, typename T::template inject_constant_t<4>, (i + 1) / 2»;
03632             using _4pm1 = typename FractionField<T>::template
      sub_t<_4p, typename FractionField<T>::one>;
03633             using dividend =
03634                 typename FractionField<T>::template mul_t<
03635                     _4p,
03636                     typename FractionField<T>::template mul_t<
03637                         _4pm1,
03638                         bernoulli_t<T, (i + 1)»>::type;
03639         public:
03640             using type = typename FractionField<T>::template div_t<dividend,
03641                 FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
03642         };
03643
03644         template<typename T, size_t i>
03645         struct tanh_coeff {
03646             using type = typename tanh_coeff_helper<T, i>::type;
03647         };
03648     }  // namespace internal
03649
03653     template<typename Integers, size_t deg>
03654     using exp = taylor<Integers, internal::exp_coeff, deg>;
03655
03659     template<typename Integers, size_t deg>
03660     using expm1 = typename polynomial<FractionField<Integers>>::template sub_t<
03661         exp<Integers, deg>,
03662         typename polynomial<FractionField<Integers>>::one>;
03663
03667     template<typename Integers, size_t deg>
03668     using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
03669
03673     template<typename Integers, size_t deg>
03674     using atan = taylor<Integers, internal::atan_coeff, deg>;
03675
03679     template<typename Integers, size_t deg>
03680     using sin = taylor<Integers, internal::sin_coeff, deg>;
03681
03685     template<typename Integers, size_t deg>
03686     using sinh = taylor<Integers, internal::sh_coeff, deg>;
03687
03692     template<typename Integers, size_t deg>
03693     using cosh = taylor<Integers, internal::cosh_coeff, deg>;
03694
03699     template<typename Integers, size_t deg>
03700     using cos = taylor<Integers, internal::cos_coeff, deg>;
03701
03706     template<typename Integers, size_t deg>
03707     using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
03708
03713     template<typename Integers, size_t deg>
```

```
03714     using asin = taylor<Integers, internal::asin_coeff, deg>;
03715
03720     template<typename Integers, size_t deg>
03721     using asinh = taylor<Integers, internal::asinh_coeff, deg>;
03722
03727     template<typename Integers, size_t deg>
03728     using atanh = taylor<Integers, internal::atanh_coeff, deg>;
03729
03734     template<typename Integers, size_t deg>
03735     using tan = taylor<Integers, internal::tan_coeff, deg>;
03736
03741     template<typename Integers, size_t deg>
03742     using tanh = taylor<Integers, internal::tanh_coeff, deg>;
03743 }  // namespace aerobus
03744
03745 // continued fractions
03746 namespace aerobus {
03749     template<int64_t... values>
03750     struct ContinuedFraction {};
03751
03754     template<int64_t a0>
03755     struct ContinuedFraction<a0> {
03757         using type = typename q64::template inject_constant_t<a0>;
03759         static constexpr double val = static_cast<double>(a0);
03760     };
03761
03765     template<int64_t a0, int64_t... rest>
03766     struct ContinuedFraction<a0, rest...> {
03768         using type = q64::template add_t<
03769                 typename q64::template inject_constant_t<a0>,
03770                 typename q64::template div_t<
03771                     typename q64::one,
03772                     typename ContinuedFraction<rest...>::type
03773                 >;
03774
03776         static constexpr double val = type::template get<double>();
03777     };
03778
03782     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>;
03784     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>;
03786     using SQRT2_fraction =
      ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
03788     using SQRT3_fraction =
      ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2>;
      // NOLINT
03789 }  // namespace aerobus
03790
03791 // known polynomials
03792 namespace aerobus {
03793     // CChebyshev
03794     namespace internal {
03795         template<int kind, size_t deg, typename I>
03796         struct chebyshev_helper {
03797             using type = typename polynomial<I>::template sub_t<
03798                 typename polynomial<I>::template mul_t<
03799                     typename polynomial<I>::template mul_t<
03800                         typename polynomial<I>::template inject_constant_t<2>,
03801                         typename polynomial<I>::X>,
03802                     typename chebyshev_helper<kind, deg - 1, I>::type
03803                 >,
03804                 typename chebyshev_helper<kind, deg - 2, I>::type
03805             >;
03806         };
03807
03808         template<typename I>
03809         struct chebyshev_helper<1, 0, I> {
03810             using type = typename polynomial<I>::one;
03811         };
03812
03813         template<typename I>
03814         struct chebyshev_helper<1, 1, I> {
03815             using type = typename polynomial<I>::X;
03816         };
03817
03818         template<typename I>
03819         struct chebyshev_helper<2, 0, I> {
03820             using type = typename polynomial<I>::one;
03821         };
03822
03823         template<typename I>
03824         struct chebyshev_helper<2, 1, I> {
03825             using type = typename polynomial<I>::template mul_t<
03826                 typename polynomial<I>::template inject_constant_t<2>,
03827                 typename polynomial<I>::X>;
03828         };
```

```
03829     }  // namespace internal
03830
03831     // Laguerre
03832     namespace internal {
03833         template<size_t deg, typename I>
03834         struct laguerre_helper {
03835             using Q = FractionField<I>;
03836             using PQ = polynomial<Q>;
03837
03838          private:
03839             // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)Lkm2)
03840             using lnm2 = typename laguerre_helper<deg - 2, I>::type;
03841             using lnm1 = typename laguerre_helper<deg - 1, I>::type;
03842             // -x + 2k-1
03843             using p = typename PQ::template val<
03844                 typename Q::template inject_constant_t<-1>,
03845                 typename Q::template inject_constant_t<2 * deg - 1»;
03846             // 1/n
03847             using factor = typename PQ::template inject_ring_t<
03848                 typename Q::template val<typename I::one, typename I::template
      inject_constant_t<deg»>;
03849
03850          public:
03851             using type = typename PQ::template mul_t <
03852                 factor,
03853                 typename PQ::template sub_t<
03854                     typename PQ::template mul_t<
03855                         p,
03856                         lnm1
03857                     >,
03858                     typename PQ::template mul_t<
03859                         typename PQ::template inject_constant_t<deg-1>,
03860                         lnm2
03861                     >
03862                 >
03863             >;
03864         };
03865
03866         template<typename I>
03867         struct laguerre_helper<0, I> {
03868             using type = typename polynomial<FractionField<I»::one;
03869         };
03870
03871         template<typename I>
03872         struct laguerre_helper<1, I> {
03873          private:
03874             using PQ = polynomial<FractionField<I»;
03875          public:
03876             using type = typename PQ::template sub_t<typename PQ::one, typename PQ::X>;
03877         };
03878     }  // namespace internal
03879
03880     // Bernstein
03881     namespace internal {
03882         template<size_t i, size_t m, typename I, typename E = void>
03883         struct bernstein_helper {};
03884
03885         template<typename I>
03886         struct bernstein_helper<0, 0, I> {
03887             using type = typename polynomial<I>::one;
03888         };
03889
03890         template<size_t i, size_t m, typename I>
03891         struct bernstein_helper<i, m, I, std::enable_if_t<
03892                     (m > 0) && (i == 0)» {
03893          private:
03894             using P = polynomial<I>;
03895          public:
03896             using type = typename P::template mul_t<
03897                     typename P::template sub_t<typename P::one, typename P::X>,
03898                     typename bernstein_helper<i, m-1, I>::type>;
03899         };
03900
03901         template<size_t i, size_t m, typename I>
03902         struct bernstein_helper<i, m, I, std::enable_if_t<
03903                     (m > 0) && (i == m)» {
03904          private:
03905             using P = polynomial<I>;
03906          public:
03907             using type = typename P::template mul_t<
03908                     typename P::X,
03909                     typename bernstein_helper<i-1, m-1, I>::type>;
03910         };
03911
03912         template<size_t i, size_t m, typename I>
03913         struct bernstein_helper<i, m, I, std::enable_if_t<
03914                     (m > 0) && (i > 0) && (i < m)» {
```

```
03915          private:
03916             using P = polynomial<I>;
03917          public:
03918             using type = typename P::template add_t<
03919                     typename P::template mul_t<
03920                         typename P::template sub_t<typename P::one, typename P::X>,
03921                         typename bernstein_helper<i, m-1, I>::type>,
03922                     typename P::template mul_t<
03923                         typename P::X,
03924                         typename bernstein_helper<i-1, m-1, I>::type»;
03925         };
03926     }  // namespace internal
03927
03928     // AllOne polynomials
03929     namespace internal {
03930         template<size_t deg, typename I>
03931         struct AllOneHelper {
03932             using type = aerobus::add_t<
03933                 typename polynomial<I>::one,
03934                 typename aerobus::mul_t<
03935                     typename polynomial<I>::X,
03936                     typename AllOneHelper<deg-1, I>::type
03937                 »;
03938         };
03939
03940         template<typename I>
03941         struct AllOneHelper<0, I> {
03942             using type = typename polynomial<I>::one;
03943         };
03944     }  // namespace internal
03945
03946     // Bessel polynomials
03947     namespace internal {
03948         template<size_t deg, typename I>
03949         struct BesselHelper {
03950          private:
03951             using P = polynomial<I>;
03952             using factor = typename P::template monomial_t<
03953                 typename I::template inject_constant_t<(2*deg - 1)>,
03954                 1>;
03955          public:
03956             using type = typename P::template add_t<
03957                 typename P::template mul_t<
03958                     factor,
03959                     typename BesselHelper<deg-1, I>::type
03960                 >,
03961                 typename BesselHelper<deg-2, I>::type
03962             >;
03963         };
03964
03965         template<typename I>
03966         struct BesselHelper<0, I> {
03967             using type = typename polynomial<I>::one;
03968         };
03969
03970         template<typename I>
03971         struct BesselHelper<1, I> {
03972          private:
03973             using P = polynomial<I>;
03974          public:
03975             using type = typename P::template add_t<
03976                 typename P::one,
03977                 typename P::X
03978             >;
03979         };
03980     }  // namespace internal
03981
03982     namespace known_polynomials {
03984         enum hermite_kind {
03986             probabilist,
03988             physicist
03989         };
03990     }
03991
03992     // hermite
03993     namespace internal {
03994         template<size_t deg, known_polynomials::hermite_kind kind, typename I>
03995         struct hermite_helper {};
03996
03997         template<size_t deg, typename I>
03998         struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist, I> {
03999          private:
04000             using hnm1 = typename hermite_helper<deg - 1,
      known_polynomials::hermite_kind::probabilist, I>::type;
04001             using hnm2 = typename hermite_helper<deg - 2,
      known_polynomials::hermite_kind::probabilist, I>::type;
04002
```

```
04003         public:
04004             using type = typename polynomial<I>::template sub_t<
04005                 typename polynomial<I>::template mul_t<typename polynomial<I>::X, hnm1>,
04006                 typename polynomial<I>::template mul_t<
04007                     typename polynomial<I>::template inject_constant_t<deg - 1>,
04008                     hnm2
04009                 >
04010             >;
04011         };
04012 
04013         template<size_t deg, typename I>
04014         struct hermite_helper<deg, known_polynomials::hermite_kind::physicist, I> {
04015         private:
04016             using hnm1 = typename hermite_helper<deg - 1, known_polynomials::hermite_kind::physicist,
      I>::type;
04017             using hnm2 = typename hermite_helper<deg - 2, known_polynomials::hermite_kind::physicist,
      I>::type;
04018 
04019         public:
04020             using type = typename polynomial<I>::template sub_t<
04021                 // 2X Hn-1
04022                 typename polynomial<I>::template mul_t<
04023                     typename pi64::val<typename I::template inject_constant_t<2>,
04024                     typename I::zero>, hnm1>,
04025 
04026                 typename polynomial<I>::template mul_t<
04027                     typename polynomial<I>::template inject_constant_t<2*(deg - 1)>,
04028                     hnm2
04029                 >
04030             >;
04031         };
04032 
04033         template<typename I>
04034         struct hermite_helper<0, known_polynomials::hermite_kind::probabilist, I> {
04035             using type = typename polynomial<I>::one;
04036         };
04037 
04038         template<typename I>
04039         struct hermite_helper<1, known_polynomials::hermite_kind::probabilist, I> {
04040             using type = typename polynomial<I>::X;
04041         };
04042 
04043         template<typename I>
04044         struct hermite_helper<0, known_polynomials::hermite_kind::physicist, I> {
04045             using type = typename pi64::one;
04046         };
04047 
04048         template<typename I>
04049         struct hermite_helper<1, known_polynomials::hermite_kind::physicist, I> {
04050             // 2X
04051             using type = typename polynomial<I>::template val<
04052                 typename I::template inject_constant_t<2>,
04053                 typename I::zero>;
04054         };
04055     }  // namespace internal
04056 
04057     // legendre
04058     namespace internal {
04059         template<size_t n, typename I>
04060         struct legendre_helper {
04061         private:
04062             using Q = FractionField<I>;
04063             using PQ = polynomial<Q>;
04064             // 1/n constant
04065             // (2n-1)/n X
04066             using fact_left = typename PQ::template monomial_t<
04067                 makefraction_t<I,
04068                     typename I::template inject_constant_t<2*n-1>,
04069                     typename I::template inject_constant_t<n>
04070                 >,
04071             1>;
04072             // (n-1) / n
04073             using fact_right = typename PQ::template val<
04074                 makefraction_t<I,
04075                     typename I::template inject_constant_t<n-1>,
04076                     typename I::template inject_constant_t<n»>;
04077 
04078         public:
04079             using type = PQ::template sub_t<
04080                     typename PQ::template mul_t<
04081                         fact_left,
04082                         typename legendre_helper<n-1, I>::type
04083                     >,
04084                     typename PQ::template mul_t<
04085                         fact_right,
04086                         typename legendre_helper<n-2, I>::type
04087                     >
```

```
04088                 >;
04089         };
04090
04091         template<typename I>
04092         struct legendre_helper<0, I> {
04093             using type = typename polynomial<FractionField<I»::one;
04094         };
04095
04096         template<typename I>
04097         struct legendre_helper<1, I> {
04098             using type = typename polynomial<FractionField<I»::X;
04099         };
04100     }  // namespace internal
04101
04102     // bernoulli polynomials
04103     namespace internal {
04104         template<size_t n>
04105         struct bernoulli_coeff {
04106             template<typename T, size_t i>
04107             struct inner {
04108              private:
04109                 using F = FractionField<T>;
04110              public:
04111                 using type = typename F::template mul_t<
04112                     typename F::template inject_ring_t<combination_t<T, i, n»,
04113                     bernoulli_t<T, n-i>
04114                 >;
04115             };
04116         };
04117     }  // namespace internal
04118
04119     namespace internal {
04120         template<size_t n>
04121         struct touchard_coeff {
04122             template<typename T, size_t i>
04123             struct inner {
04124                 using type = stirling_2_t<T, n, i>;
04125             };
04126         };
04127     }  // namespace internal
04128
04129     namespace internal {
04130         template<typename I = aerobus::i64>
04131         struct AbelHelper {
04132          private:
04133             using P = aerobus::polynomial<I>;
04134
04135          public:
04136             // to keep recursion working, we need to operate on a*n and not just a
04137             template<size_t deg, I::inner_type an>
04138             struct Inner {
04139                 // abel(n, a) = (x-an) * abel(n-1, a)
04140                 using type = typename aerobus::mul_t<
04141                     typename Inner<deg-1, an>::type,
04142                     typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
04143                 >;
04144             };
04145
04146             // abel(0, a) = 1
04147             template<I::inner_type an>
04148             struct Inner<0, an> {
04149                 using type = P::one;
04150             };
04151
04152             // abel(1, a) = X
04153             template<I::inner_type an>
04154             struct Inner<1, an> {
04155                 using type = P::X;
04156             };
04157         };
04158     }  // namespace internal
04159
04161     namespace known_polynomials {
04162
04171         template<size_t n, auto a, typename I = aerobus::i64>
04172         using abel = typename internal::AbelHelper<I>::template Inner<n, a*n>::type;
04173
04181         template <size_t deg, typename I = aerobus::i64>
04182         using chebyshev_T = typename internal::chebyshev_helper<1, deg, I>::type;
04183
04193         template <size_t deg, typename I = aerobus::i64>
04194         using chebyshev_U = typename internal::chebyshev_helper<2, deg, I>::type;
04195
04205         template <size_t deg, typename I = aerobus::i64>
04206         using laguerre = typename internal::laguerre_helper<deg, I>::type;
04207
04214         template <size_t deg, typename I = aerobus::i64>
```

```
04215         using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist,
      I>::type;
04216
04223         template <size_t deg, typename I = aerobus::i64>
04224         using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist, I>::type;
04225
04236         template<size_t i, size_t m, typename I = aerobus::i64>
04237         using bernstein = typename internal::bernstein_helper<i, m, I>::type;
04238
04248         template<size_t deg, typename I = aerobus::i64>
04249         using legendre = typename internal::legendre_helper<deg, I>::type;
04250
04260         template<size_t deg, typename I = aerobus::i64>
04261         using bernoulli = taylor<I, internal::bernoulli_coeff<deg>::template inner, deg>;
04262
04269         template<size_t deg, typename I = aerobus::i64>
04270         using allone = typename internal::AllOneHelper<deg, I>::type;
04271
04279         template<size_t deg, typename I = aerobus::i64>
04280         using bessel = typename internal::BesselHelper<deg, I>::type;
04281
04289         template<size_t deg, typename I = aerobus::i64>
04290         using touchard = taylor<I, internal::touchard_coeff<deg>::template inner, deg>;
04291     }  // namespace known_polynomials
04292 }  // namespace aerobus
04293
04294 namespace aerobus {
04295     namespace libm {
04296         namespace internal {
04297             template<typename T>
04298             struct exp2_poly;
04299
04300             template<>
04301             struct exp2_poly<double> {
04302                 using type = aerobus::polynomial<aerobus::q64>::template val<
04303                     aerobus::make_q64_t<388, 10641171255427>,
04304                     aerobus::make_q64_t<2296, 5579977897299>,
04305                     aerobus::make_q64_t<4963, 697799188641>,
04306                     aerobus::make_q64_t<15551, 152884735543>,
04307                     aerobus::make_q64_t<21273, 16096444733>,
04308                     aerobus::make_q64_t<683500, 44811710339>,
04309                     aerobus::make_q64_t<44397656049575, 288230376151711744>,
04310                     aerobus::make_q64_t<192156823709857, 144115188075855872>,
04311                     aerobus::make_q64_t<693059242663871, 72057594037927936>,
04312                     aerobus::make_q64_t<1999746264802375, 36028797018963968>,
04313                     aerobus::make_q64_t<4327536028902111, 18014398509481984>,
04314                     aerobus::make_q64_t<6243314768165359, 9007199254740992>,
04315                     aerobus::q64::one>;
04316             };
04317
04318             template<>
04319             struct exp2_poly<float> {
04320                 using type = aerobus::polynomial<aerobus::q32>::template val<
04321                     aerobus::make_q32_t<8, 375115>,
04322                     aerobus::make_q32_t<30, 208117>,
04323                     aerobus::make_q32_t<109, 81261>,
04324                     aerobus::make_q32_t<5161841, 536870912>,
04325                     aerobus::make_q32_t<3724869, 67108864>,
04326                     aerobus::make_q32_t<16121323, 67108864>,
04327                     aerobus::make_q32_t<1453635, 2097152>,
04328                     aerobus::q32::one>;
04329             };
04330
04331             #ifdef WITH_CUDA_FP16
04332             template<>
04333             struct exp2_poly<__half> {
04334                 using type = aerobus::polynomial<aerobus::q16>::template val<
04335                     aerobus::make_q16_t<3, 212>,
04336                     aerobus::make_q16_t<13, 256>,
04337                     aerobus::make_q16_t<31, 128>,
04338                     aerobus::make_q16_t<1419, 2048>,
04339                     aerobus::q16::one>;
04340             };
04341
04342             template<>
04343             struct exp2_poly<__half2> {
04344                 using type = aerobus::polynomial<aerobus::q16>::template val<
04345                     aerobus::make_q16_t<3, 212>,
04346                     aerobus::make_q16_t<13, 256>,
04347                     aerobus::make_q16_t<31, 128>,
04348                     aerobus::make_q16_t<1419, 2048>,
04349                     aerobus::q16::one>;
04350             };
04351             #endif
04352
04353             template<typename P>
04354             struct sin_poly;
```

```
04355
04356             template<>
04357             struct sin_poly<double> {
04358                 // approximates sin(x)/x over [-pi/4, pi/4] with precision 9.318608669702093e-20
04359                 using type = typename aerobus::polynomial<aerobus::q64>::simplify_t<
04360                     typename aerobus::polynomial<aerobus::q64>:: template val<
04361                         aerobus::make_q64_t<-43, 1042171195712159>,
04362                         aerobus::make_q64_t<-89, 136637767615782>,
04363                         aerobus::make_q64_t<123, 766493207966>,
04364                         aerobus::make_q64_t<-18133, 723813242548>,
04365                         aerobus::make_q64_t<122341, 44395102410>,
04366                         aerobus::make_q64_t<-11252871, 56714469841>,
04367                         aerobus::make_q64_t<2401919801264179, 288230376151711744>,
04368                         aerobus::make_q64_t<-6004799503160661, 36028797018963968>,
04369                         aerobus::q64::one»;
04370             };
04371
04372             template<>
04373             struct sin_poly<float> {
04374                 // approximates sin(x)/x over [-pi/4, pi/4] with precision 1.941356e-10
04375                 // must be evaluated in x*x as we removed half the coefficients to have a dense
      polynomial
04376                 using type = typename aerobus::polynomial<aerobus::q32>::simplify_t<
04377                     typename aerobus::polynomial<aerobus::q32>:: template val<
04378                         aerobus::make_q32_t<1, 357073>,
04379                         aerobus::make_q32_t<-67, 337533>,
04380                         aerobus::make_q32_t<4473945, 536870912>,
04381                         aerobus::make_q32_t<-11184811, 67108864>,
04382                         aerobus::q32::one»;
04383             };
04384
04385             #ifdef WITH_CUDA_FP16
04386             template<>
04387             struct sin_poly<__half> {
04388                 // approximates sin(x)/x over [-pi/4, pi/4] with precision 6.389e-6
04389                 // must be evaluated in x*x as we removed half the coefficients to have a dense
      polynomial
04390                 using type = typename aerobus::polynomial<aerobus::q16>::simplify_t<
04391                     typename aerobus::polynomial<aerobus::q16>:: template val<
04392                         aerobus::make_q16_t<1, 123>,
04393                         aerobus::make_q16_t<-1365, 8192>,
04394                         aerobus::q16::one»;
04395             };
04396
04397             template<>
04398             struct sin_poly<__half2> {
04399                 // approximates sin(x)/x over [-pi/4, pi/4] with precision 6.389e-6
04400                 // must be evaluated in x*x as we removed half the coefficients to have a dense
      polynomial
04401                 using type = typename aerobus::polynomial<aerobus::q16>::simplify_t<
04402                     typename aerobus::polynomial<aerobus::q16>:: template val<
04403                         aerobus::make_q16_t<1, 123>,
04404                         aerobus::make_q16_t<-1365, 8192>,
04405                         aerobus::q16::one»;
04406             };
04407             #endif
04408
04409             template<typename T>
04410             struct cos_poly;
04411
04412             template <>
04413             struct cos_poly<double> {
04414                 // approximates (cos(x) - 1 + x^2)/x^4
04415                 // must be evaluated in x*x as we removed half the coefficients to have a dense
      polynomial
04416                 using type = typename aerobus::polynomial<aerobus::q64>::simplify_t<
04417                     typename aerobus::polynomial<aerobus::q64>:: template val<
04418                         aerobus::make_q64_t<14, 407833194230757>,
04419                         aerobus::make_q64_t<-283, 24728864074278>,
04420                         aerobus::make_q64_t<136, 65144855767>,
04421                         aerobus::make_q64_t<-4089, 14838163607>,
04422                         aerobus::make_q64_t<452396, 18240606721>,
04423                         aerobus::make_q64_t<-6405119470037555, 4611686018427387904>,
04424                         aerobus::make_q64_t<6004799503160661, 144115188075855872>»;
04425             };
04426
04427             template <>
04428             struct cos_poly<float> {
04429                 // approximates (cos(x) - 1 + x^2)/x^4
04430                 // must be evaluated in x*x as we removed half the coefficients to have a dense
      polynomial
04431                 using type = typename aerobus::polynomial<aerobus::q32>::simplify_t<
04432                     typename aerobus::polynomial<aerobus::q32>:: template val<
04433                         aerobus::make_q32_t<-1, 3112816>,
04434                         aerobus::make_q32_t<1, 40237>,
04435                         aerobus::make_q32_t<-119, 85679>,
04436                         aerobus::make_q32_t<11184811, 268435456>»;
```

```
04437             };
04438
04439 #ifdef WITH_CUDA_FP16
04440             template<>
04441             struct cos_poly<__half> {
04442                 // approximates (cos(x) - 1 + x^2)/x^4
04443                 // must be evaluated in x*x as we removed half the coefficients to have a dense
      polynomial
04444                 using type = typename aerobus::polynomial<aerobus::q16>::simplify_t<
04445                     typename aerobus::polynomial<aerobus::q16>::template val<
04446                         aerobus::make_q16_t<-1, 16321>,
04447                         aerobus::make_q16_t<-1, 756>,
04448                         aerobus::make_q16_t<1, 24»>;
04449             };
04450
04451             template<>
04452             struct cos_poly<__half2> {
04453                 // approximates (cos(x) - 1 + x^2)/x^4
04454                 // must be evaluated in x*x as we removed half the coefficients to have a dense
      polynomial
04455                 using type = typename aerobus::polynomial<aerobus::q16>::simplify_t<
04456                     typename aerobus::polynomial<aerobus::q16>::template val<
04457                         aerobus::make_q16_t<-1, 16321>,
04458                         aerobus::make_q16_t<-1, 756>,
04459                         aerobus::make_q16_t<1, 24»>;
04460             };
04461             #endif
04462         }  // namespace internal
04463
04468         template<typename T>
04469         static T exp2(const T&x) {
04470             using poly = internal::exp2_poly<T>::type;
04471             if (x >= (aerobus::internal::arithmetic_helpers<T>::zero) &&
04472                 x < (aerobus::internal::arithmetic_helpers<T>::one)) {
04473                     return poly::eval(x);
04474             } else {
04475                 // TODO(JeWaVe): handle denormals
04476                 auto i = static_cast<typename aerobus::internal::arithmetic_helpers<T>::integers>(
04477                     aerobus::meta_libm<T>::floor(x));
04478                 T eps = x - i;
04479                 T mantissa = poly::eval(eps);
04480                 uint64_t* p = reinterpret_cast<uint64_t*>(&mantissa);
04481                 uint64_t exponent = (*p » 52) & 0x7FF;
04482                 exponent += i;
04483                 *p &= ~(0x7FFULL « 52);
04484                 *p |= (exponent & 0x7FF) « 52;
04485                 return mantissa;
04486             }
04487         }
04488
04489         template<typename T>
04490         static T cos(const T& x);
04491
04499         template<typename T>
04500         static T sin(const T& x) {
04501             using upper_type = aerobus::internal::arithmetic_helpers<T>::upper_type;
04502             using constants = aerobus::internal::arithmetic_helpers<upper_type>;
04503             using poly = internal::sin_poly<T>::type;
04504             constexpr upper_type zero = constants::zero;
04505             const upper_type pi_4 = constants::pi_4;
04506             const upper_type pi = constants::pi;
04507             const upper_type two_pi = constants::two_pi;
04508             const upper_type pi_2 = constants::pi_2;
04509             upper_type X = static_cast<upper_type>(x);
04510             // TODO(JeWaVe): infinities
04511             if (x == 0 || x == -0) {
04512                 return x;
04513             } else if (x < zero) {
04514                 return -sin(static_cast<T>(-X));
04515             } else if (x <= std::numeric_limits<T>::epsilon() || std::isnan(x)) {
04516                 return x;
04517             } else if (X > pi_4 && X <= pi_2) {
04518                 return aerobus::libm::cos(static_cast<T>(pi_2 - X));
04519             } else if (X >= pi_2 && X <= pi) {
04520                 return sin(static_cast<T>(pi - X));
04521             } else if (X > pi && X <= two_pi) {
04522                 return -sin(static_cast<T>(X - pi));
04523             } else if (X > two_pi) {
04524                 T i = static_cast<T>(aerobus::meta_libm<upper_type>::fmod(X, two_pi));
04525                 return sin(i);
04526             } else {
04527                 return x * poly::eval(x*x);
04528             }
04529         }
04530
04531
04539         template<typename T>
```

```
04540         static T cos(const T& x) {
04541             using poly = internal::cos_poly<T>::type;
04542             using upper_type = aerobus::internal::arithmetic_helpers<T>::upper_type;
04543             using constants = aerobus::internal::arithmetic_helpers<upper_type>;
04544             constexpr upper_type zero = constants::zero;
04545             constexpr upper_type half = constants::half;
04546             constexpr upper_type one = constants::one;
04547             constexpr upper_type pi_4 = constants::pi_4;
04548             constexpr upper_type pi = constants::pi;
04549             constexpr upper_type two_pi = constants::two_pi;
04550             constexpr upper_type pi_2 = constants::pi_2;
04551             upper_type X = static_cast<upper_type>(x);
04552             // domain reduction
04553             if (x == 0 || x == -0) {
04554                 return aerobus::internal::arithmetic_helpers<T>::one;
04555             } else if (x < zero) {
04556                 return cos(static_cast<T>(-X));
04557             } else if (std::isnan(x)) {
04558                 return x;
04559             } else if (x <= std::numeric_limits<T>::epsilon()) {
04560                 return constants::one;
04561             } else if (X > pi_4 && X <= pi_2) {
04562                 return aerobus::libm::sin(static_cast<T>(pi_2 - X));
04563             } else if (X >= pi_2 && X < pi) {
04564                 return -cos(static_cast<T>(X - pi));
04565             } else if (X >= pi && X < two_pi) {
04566                 return -cos(static_cast<T>(pi - X));
04567             } else if (X >= two_pi) {
04568                 T i = static_cast<T>(aerobus::meta_libm<upper_type>::fmod(X, two_pi));
04569                 return cos(i);
04570             } else {
04571                 T x2 = x*x;
04572                 T x4 = x2*x2;
04573                 return x4 * poly::eval(x2) + one - x2*half;
04574             }
04575         }
04576     }  // namespace libm
04577 }  // namespace aerobus
04578
04579 #ifdef AEROBUS_CONWAY_IMPORTS
04580
04581 // conway polynomials
04582 namespace aerobus {
04586     template<int p, int n>
04587     struct ConwayPolynomial {};
04588
04589 #ifndef DO_NOT_DOCUMENT
04590     #define ZPZV ZPZ::template val
04591     #define POLYV aerobus::polynomial<ZPZ>::template val
04592     template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<1>>; };  // NOLINT
04593     template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1>>; }; // NOLINT
04594     template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
04595     template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
04596     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
04597     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
04598     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
04599     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
04600     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
04601     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
04602     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
04603     template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>,
      ZPZV<0>, ZPZV<1>, ZPZV<1>>; };  // NOLINT
04604     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
04605     template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>,
      ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
04606     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<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>>; };  // NOLINT
04607     template<> struct ConwayPolynomial<2, 16> { using ZPZ = aerobus::zpz<2>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; };  // NOLINT
04608     template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; };  // NOLINT
04609     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
04610     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<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; };  //
      NOLINT
04611     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
04612     template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<1»; };  // NOLINT
04613     template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; };  // NOLINT
04614     template<> struct ConwayPolynomial<3, 3> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1»; };  // NOLINT
04615     template<> struct ConwayPolynomial<3, 4> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; };  // NOLINT
04616     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
04617     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
04618     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
04619     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
04620     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
04621     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
04622     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<2>,
      ZPZV<0>, ZPZV<1»; };  // NOLINT
04623     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
04624     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
04625     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
04626     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<2>, ZPZV<0>, ZPZV<0>,
      ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1»; };  // NOLINT
04627     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<2>,
      ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<2»; };  // NOLINT
04628     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
04629     template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>,
      ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2»; };  // NOLINT
04630     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<2>, ZPZV<0>, ZPZV<1»; };  //
      NOLINT
04631     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
04632     template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<3»; };  // NOLINT
04633     template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; };  // NOLINT
04634     template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; };  // NOLINT
04635     template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; };  // NOLINT
04636     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
04637     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
04638     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
04639     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
04640     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
04641     template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<4>, ZPZV<1>,
      ZPZV<2>>; };  // NOLINT
04642     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
04643     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
04644     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
04645     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
04646     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
04647     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
04648     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
04649     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
04650     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
04651     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
04652     template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<4>>; };  // NOLINT
04653     template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3>>; };  // NOLINT
04654     template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4>>; };  // NOLINT
04655     template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3>>; };  // NOLINT
04656     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
04657     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
04658     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
04659     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
04660     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
04661     template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
      ZPZV<3>>; };  // NOLINT
04662     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
04663     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
04664     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
04665     template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<6>,
      ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3>>; };  // NOLINT
04666     template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>,
      ZPZV<6>, ZPZV<6>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<4>>; };  // NOLINT
04667     template<> struct ConwayPolynomial<7, 16> { using ZPZ = aerobus::zpz<7>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<5>,
      ZPZV<3>, ZPZV<4>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<4>, ZPZV<3>>; };  // NOLINT
04668     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
04669     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
04670     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
```

```
04671     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
04672     template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<9»; };  // NOLINT
04673     template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; };  // NOLINT
04674     template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9»; };  // NOLINT
04675     template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; };  // NOLINT
04676     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
04677     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
04678     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
04679     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
04680     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
04681     template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
      ZPZV<2»; };  // NOLINT
04682     template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<10>, ZPZV<9»; };  // NOLINT
04683     template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<4>, ZPZV<2>, ZPZV<5>, ZPZV<5>,
      ZPZV<6>, ZPZV<5>, ZPZV<2»; };  // NOLINT
04684     template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<9»; };  // NOLINT
04685     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<2>, ZPZV<9>, ZPZV<6>,
      ZPZV<4>, ZPZV<8>, ZPZV<6>, ZPZV<10>, ZPZV<2»; };  // NOLINT
04686     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<10>,
      ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<0>, ZPZV<9»; };  // NOLINT
04687     template<> struct ConwayPolynomial<11, 16> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>,
      ZPZV<1>, ZPZV<3>, ZPZV<5>, ZPZV<3>, ZPZV<10>, ZPZV<9>, ZPZV<2»; };  // NOLINT
04688     template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9»; };  // NOLINT
04689     template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<10>, ZPZV<8>,
      ZPZV<3>, ZPZV<9>, ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<9>, ZPZV<8>, ZPZV<2>, ZPZV<2»; };  // NOLINT
04690     template<> struct ConwayPolynomial<11, 19> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<2>, ZPZV<9»; };  //
      NOLINT
04691     template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>,
      ZPZV<9>, ZPZV<1>, ZPZV<5>, ZPZV<7>, ZPZV<2>, ZPZV<4>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>, ZPZV<2»; };
      // NOLINT
04692     template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<11»; };  // NOLINT
04693     template<> struct ConwayPolynomial<13, 2> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; };  // NOLINT
04694     template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; };  // NOLINT
04695     template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; };  // NOLINT
04696     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
04697     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
04698     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
04699     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
04700     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
04701     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
04702     template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<3>, ZPZV<11»; };  // NOLINT
04703     template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<11>, ZPZV<3>, ZPZV<1>,
      ZPZV<1>, ZPZV<4>, ZPZV<2»; };  // NOLINT
04704     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<12>, ZPZV<11>>; };  // NOLINT
04705     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
04706     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
04707     template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<12>,
      ZPZV<8>, ZPZV<2>, ZPZV<12>, ZPZV<9>, ZPZV<12>, ZPZV<6>, ZPZV<2>>; };  // NOLINT
04708     template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<6>, ZPZV<11>>; };  // NOLINT
04709     template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<4>, ZPZV<11>,
      ZPZV<11>, ZPZV<9>, ZPZV<5>, ZPZV<3>, ZPZV<5>, ZPZV<6>, ZPZV<0>, ZPZV<9>, ZPZV<2>>; };  // NOLINT
04710     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
04711     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<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
04712     template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<14>>; };  // NOLINT
04713     template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3>>; };  // NOLINT
04714     template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14>>; };  // NOLINT
04715     template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<3>>; };  // NOLINT
04716     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
04717     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
04718     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
04719     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
04720     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
04721     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
04722     template<> struct ConwayPolynomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<5>, ZPZV<14>>; };  // NOLINT
04723     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
04724     template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<14>>; };  // NOLINT
04725     template<> struct ConwayPolynomial<17, 14> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<1>, ZPZV<8>,
      ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3>>; };  // NOLINT
04726     template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>,
      ZPZV<4>, ZPZV<16>, ZPZV<6>, ZPZV<14>, ZPZV<14>, ZPZV<14>>; };  // NOLINT
04727     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<1>, ZPZV<13>,
      ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3>>; };  // NOLINT
04728     template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<14>>; };  // NOLINT
04729     template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<16>,
      ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3>>; };  // NOLINT
04730     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
04731     template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>,
      ZPZV<16>, ZPZV<14>, ZPZV<13>, ZPZV<3>, ZPZV<14>, ZPZV<9>, ZPZV<1>, ZPZV<13>, ZPZV<2>, ZPZV<5>,
      ZPZV<3>>; };  // NOLINT
04732     template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
      POLYV<ZPZV<1>, ZPZV<17>>; };  // NOLINT
04733     template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
      POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2>>; };  // NOLINT
04734     template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17>>; };  // NOLINT
04735     template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2>>; };  // NOLINT
```

```
04736     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
04737     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
04738     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
04739     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
04740     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
04741     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
04742     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<8>, ZPZV<17»; };  // NOLINT
04743     template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<18>, ZPZV<2>, ZPZV<9>,
      ZPZV<16>, ZPZV<7>, ZPZV<2»; };  // NOLINT
04744     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<11>, ZPZV<17»; };  // NOLINT
04745     template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<11>, ZPZV<11>,
      ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<7>, ZPZV<2»; };  // NOLINT
04746     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<1>, ZPZV<10>,
      ZPZV<11>, ZPZV<13>, ZPZV<15>, ZPZV<14>, ZPZV<0>, ZPZV<17»; };  // NOLINT
04747     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<12>,
      ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<9>, ZPZV<6>, ZPZV<14>, ZPZV<2»; };  // NOLINT
04748     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<2>, ZPZV<17»; };  // NOLINT
04749     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<10>, ZPZV<9>, ZPZV<7>,
      ZPZV<17>, ZPZV<5>, ZPZV<0>, ZPZV<16>, ZPZV<5>, ZPZV<7>, ZPZV<3>, ZPZV<14>, ZPZV<2»; };  // NOLINT
04750     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<18>, ZPZV<17»; };  //
      NOLINT
04751     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<16>,
      ZPZV<13>, ZPZV<0>, ZPZV<4>, ZPZV<7>, ZPZV<8>, ZPZV<6>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<11>, ZPZV<2»;
      };  // NOLINT
04752     template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<18»; };  // NOLINT
04753     template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; };  // NOLINT
04754     template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; };  // NOLINT
04755     template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; };  // NOLINT
04756     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
04757     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
04758     template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18»; };  // NOLINT
04759     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
04760     template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
      // NOLINT
04761     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
04762     template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>,
      ZPZV<7>, ZPZV<18»; };  // NOLINT
04763     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
04764     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
04765     template<> struct ConwayPolynomial<23, 14> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<16>, ZPZV<1>,
      ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5»; };  // NOLINT
04766     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<2>,
      ZPZV<8>, ZPZV<15>, ZPZV<9>, ZPZV<7>, ZPZV<18>, ZPZV<18»; };  // NOLINT
04767     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<19>,
      ZPZV<19>, ZPZV<16>, ZPZV<13>, ZPZV<1>, ZPZV<14>, ZPZV<17>, ZPZV<5»; };  // NOLINT
04768     template<> struct ConwayPolynomial<23, 17> { using ZPZ = aerobus::zpz<23>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<18>»; };  // NOLINT
04769     template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<18>, ZPZV<2>, ZPZV<1>,
      ZPZV<18>, ZPZV<3>, ZPZV<16>, ZPZV<21>, ZPZV<0>, ZPZV<11>, ZPZV<3>, ZPZV<19>, ZPZV<5>»; };  // NOLINT
04770     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
04771     template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<27>»; };  // NOLINT
04772     template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2>»; };  // NOLINT
04773     template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27>»; };  // NOLINT
04774     template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2>»; };  // NOLINT
04775     template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27>»; };  // NOLINT
04776     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
04777     template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27>»; };  // NOLINT
04778     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
04779     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
04780     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
04781     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
04782     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
04783     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
04784     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
04785     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
04786     template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<27>,
      ZPZV<2>, ZPZV<18>, ZPZV<23>, ZPZV<1>, ZPZV<27>, ZPZV<10>, ZPZV<2>»; };  // NOLINT
04787     template<> struct ConwayPolynomial<29, 17> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27>»; };  // NOLINT
04788     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<24>, ZPZV<1>, ZPZV<1>,
      ZPZV<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<8>, ZPZV<16>, ZPZV<19>, ZPZV<14>, ZPZV<2>»; };  // NOLINT
04789     template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<27>»; };  //
      NOLINT
04790     template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<28>»; };  // NOLINT
04791     template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3>»; };  // NOLINT
04792     template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28>»; };  // NOLINT
04793     template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3>»; };  // NOLINT
04794     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
04795     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
04796     template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>»; };  // NOLINT
04797     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
04798     template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<20>, ZPZV<29>, ZPZV<28>»; };
      // NOLINT
04799     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
04800     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<20>, ZPZV<28>»; };  // NOLINT
04801     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
04802     template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<28»; };  // NOLINT
04803     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
04804     template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>,
      ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28»; };  // NOLINT
04805     template<> struct ConwayPolynomial<31, 16> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>,
      ZPZV<24>, ZPZV<26>, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<3»; };  // NOLINT
04806     template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<28»; };  // NOLINT
04807     template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, 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
04808     template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; };  //
      NOLINT
04809     template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<35»; };  // NOLINT
04810     template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; };  // NOLINT
04811     template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; };  // NOLINT
04812     template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; };  // NOLINT
04813     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
04814     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
04815     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
04816     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
04817     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
04818     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
04819     template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<2>, ZPZV<35»; };  // NOLINT
04820     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
04821     template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; };  // NOLINT
04822     template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35>, ZPZV<35>, ZPZV<1>,
      ZPZV<32>, ZPZV<16>, ZPZV<1>, ZPZV<9>, ZPZV<2»; };  // NOLINT
04823     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<1>, ZPZV<31>,
      ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<33>, ZPZV<35»; };  // NOLINT
04824     template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35»; };  // NOLINT
04825     template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<19>, ZPZV<15>,
      ZPZV<1>, ZPZV<22>, ZPZV<20>, ZPZV<12>, ZPZV<32>, ZPZV<14>, ZPZV<27>, ZPZV<20>, ZPZV<2»; };  // NOLINT
04826     template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<23>, ZPZV<35»; };  //
      NOLINT
04827     template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<35»; };  // NOLINT
04828     template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; };  // NOLINT
04829     template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35»; };  // NOLINT
04830     template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6»; };  // NOLINT
04831     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
04832     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
04833     template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; };  // NOLINT
04834     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
```

```
04835     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
04836     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
04837     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<20>, ZPZV<35»; };  // NOLINT
04838     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
04839     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<13>, ZPZV<35»; };  // NOLINT
04840     template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<15>, ZPZV<4>,
      ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6»; };  // NOLINT
04841     template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>,
      ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; };  // NOLINT
04842     template<> struct ConwayPolynomial<41, 17> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<35»; };  // NOLINT
04843     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
04844     template<> struct ConwayPolynomial<41, 19> { using ZPZ = aerobus::zpz<41>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<35»; };  //
      NOLINT
04845     template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<40»; };  // NOLINT
04846     template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; };  // NOLINT
04847     template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; };  // NOLINT
04848     template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3»; };  // NOLINT
04849     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
04850     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
04851     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
04852     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
04853     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
04854     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
04855     template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<7>, ZPZV<40»; };  // NOLINT
04856     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
04857     template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<40»; };  // NOLINT
04858     template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<38>, ZPZV<22>, ZPZV<24>,
      ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<19>, ZPZV<3»; };  // NOLINT
04859     template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<37>,
      ZPZV<22>, ZPZV<42>, ZPZV<4>, ZPZV<15>, ZPZV<37>, ZPZV<40»; };  // NOLINT
04860     template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<40»; };  // NOLINT
04861     template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, 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
04862     template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<40»; };  //
      NOLINT
04863     template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<42»; };  // NOLINT
04864     template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; };  // NOLINT
04865     template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; };  // NOLINT
04866     template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; };  // NOLINT
```

```
04867     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
04868     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
04869     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
04870     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
04871     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
04872     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
04873     template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<6>, ZPZV<42>>; };  // NOLINT
04874     template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<40>, ZPZV<35>, ZPZV<12>, ZPZV<46>,
      ZPZV<14>, ZPZV<9>, ZPZV<5>>; };  // NOLINT
04875     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<5>, ZPZV<42>>; };  // NOLINT
04876     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<36>, ZPZV<20>, ZPZV<30>,
      ZPZV<17>, ZPZV<24>, ZPZV<9>, ZPZV<32>, ZPZV<5>>; };  // NOLINT
04877     template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<43>,
      ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42>>; };  // NOLINT
04878     template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<42>>; };  // NOLINT
04879     template<> struct ConwayPolynomial<47, 18> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<41>, ZPZV<42>,
      ZPZV<26>, ZPZV<44>, ZPZV<24>, ZPZV<22>, ZPZV<11>, ZPZV<5>, ZPZV<45>, ZPZV<33>, ZPZV<5>>; };  // NOLINT
04880     template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<42>>; };  //
      NOLINT
04881     template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<51>>; };  // NOLINT
04882     template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2>>; };  // NOLINT
04883     template<> struct ConwayPolynomial<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51>>; };  // NOLINT
04884     template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2>>; };  // NOLINT
04885     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
04886     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
04887     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
04888     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
04889     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<13>, ZPZV<5>, ZPZV<51>>; };
      // NOLINT
04890     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
04891     template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<15>, ZPZV<51>>; };  // NOLINT
04892     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
04893     template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<52>, ZPZV<28>, ZPZV<51>>; };  // NOLINT
04894     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
04895     template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>,
      ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51>>; };  // NOLINT
04896     template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<51>>; };  // NOLINT
04897     template<> struct ConwayPolynomial<53, 18> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<52>, ZPZV<31>, ZPZV<51>,
      ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<11>, ZPZV<2>>; };  // NOLINT
04898     template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<51>>; };  //
      NOLINT
04899     template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<57»; };  // NOLINT
04900     template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
    POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; };  // NOLINT
04901     template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; };  // NOLINT
04902     template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; };  // NOLINT
04903     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
04904     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
04905     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
04906     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
04907     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
04908     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
04909     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<6>, ZPZV<57»; };  // NOLINT
04910     template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>,
    ZPZV<8>, ZPZV<1>, ZPZV<2»; };  // NOLINT
04911     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<3>, ZPZV<57»; };  // NOLINT
04912     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
04913     template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>,
    ZPZV<24>, ZPZV<23>, ZPZV<13>, ZPZV<39>, ZPZV<58>, ZPZV<57»; };  // NOLINT
04914     template<> struct ConwayPolynomial<59, 17> { using ZPZ = aerobus::zpz<59>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<57»; };  // NOLINT
04915     template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<38>, ZPZV<27>,
    ZPZV<11>, ZPZV<14>, ZPZV<7>, ZPZV<44>, ZPZV<16>, ZPZV<47>, ZPZV<34>, ZPZV<32>, ZPZV<2»; };  // NOLINT
04916     template<> struct ConwayPolynomial<59, 19> { using ZPZ = aerobus::zpz<59>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<57»; };  //
    NOLINT
04917     template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<59»; };  // NOLINT
04918     template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; };  // NOLINT
04919     template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; };  // NOLINT
04920     template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; };  // NOLINT
04921     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
04922     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
04923     template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; };  // NOLINT
04924     template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<56>, ZPZV<2»; };  // NOLINT
04925     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
04926     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
04927     template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<18>, ZPZV<59»; };  // NOLINT
04928     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
04929     template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<59»; };  // NOLINT
04930     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
04931     template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>,
    ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59»; };  // NOLINT
04932     template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<59»; };  // NOLINT
```

```
04933     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
04934     template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; }; //
      NOLINT
04935     template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<65»; }; // NOLINT
04936     template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
04937     template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT
04938     template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; }; // NOLINT
04939     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
04940     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
04941     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
04942     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
04943     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
04944     template<> struct ConwayPolynomial<67, 10> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<0>, ZPZV<16>, ZPZV<7>, ZPZV<23>,
      ZPZV<2»; }; // NOLINT
04945     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
04946     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
04947     template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<65»; }; // NOLINT
04948     template<> struct ConwayPolynomial<67, 14> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<22>, ZPZV<5>,
      ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2»; }; // NOLINT
04949     template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>,
      ZPZV<52>, ZPZV<41>, ZPZV<20>, ZPZV<21>, ZPZV<46>, ZPZV<65»; }; // NOLINT
04950     template<> struct ConwayPolynomial<67, 17> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<65»; }; // NOLINT
04951     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
04952     template<> struct ConwayPolynomial<67, 19> { using ZPZ = aerobus::zpz<67>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<65»; }; //
      NOLINT
04953     template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
      POLYV<ZPZV<1>, ZPZV<64»; }; // NOLINT
04954     template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
      POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT
04955     template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; // NOLINT
04956     template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; }; // NOLINT
04957     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
04958     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
04959     template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<64»; }; // NOLINT
04960     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
04961     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
04962     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
04963     template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<48>, ZPZV<64»; }; // NOLINT
04964     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
04965     template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<64»; }; // NOLINT
```

```
04966     template<> struct ConwayPolynomial<71, 15> { using ZPZ = aerobus::zpz<71>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>,
      ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64>; };  // NOLINT
04967     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<3>, ZPZV<64>; };  // NOLINT
04968     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
04969     template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<68>; };  // NOLINT
04970     template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5>; };  // NOLINT
04971     template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68>; };  // NOLINT
04972     template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5>; };  // NOLINT
04973     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
04974     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
04975     template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68>; };  // NOLINT
04976     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
04977     template<> struct ConwayPolynomial<73, 9> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<72>, ZPZV<15>, ZPZV<68>; };
      // NOLINT
04978     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
04979     template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<5>, ZPZV<68>; };  // NOLINT
04980     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
04981     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
04982     template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<10>, ZPZV<33>, ZPZV<57>, ZPZV<57>, ZPZV<62>, ZPZV<68>; };  // NOLINT
04983     template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<68>; };  // NOLINT
04984     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
04985     template<> struct ConwayPolynomial<79, 1> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<76>; };  // NOLINT
04986     template<> struct ConwayPolynomial<79, 2> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3>; };  // NOLINT
04987     template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76>; };  // NOLINT
04988     template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<66>, ZPZV<3>; };  // NOLINT
04989     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
04990     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
04991     template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76>; };  // NOLINT
04992     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
04993     template<> struct ConwayPolynomial<79, 9> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<19>, ZPZV<76>; };
      // NOLINT
04994     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
04995     template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<3>, ZPZV<76>; };  // NOLINT
04996     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
04997     template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<78>, ZPZV<4>, ZPZV<76>; };  // NOLINT
04998     template<> struct ConwayPolynomial<79, 17> { using ZPZ = aerobus::zpz<79>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<76>; };  // NOLINT
```

```
04999     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
05000     template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<81>; };  // NOLINT
05001     template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2>; };  // NOLINT
05002     template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81>; };  // NOLINT
05003     template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2>; };  // NOLINT
05004     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
05005     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
05006     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
05007     template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<2>; };  //
      NOLINT
05008     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
05009     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
05010     template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<17>, ZPZV<81>; };  // NOLINT
05011     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
05012     template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<81>; };  // NOLINT
05013     template<> struct ConwayPolynomial<83, 17> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<81>; };  // NOLINT
05014     template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<81>; };  //
      NOLINT
05015     template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<86>; };  // NOLINT
05016     template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3>; };  // NOLINT
05017     template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86>; };  // NOLINT
05018     template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3>; };  // NOLINT
05019     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
05020     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
05021     template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86>; };  // NOLINT
05022     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
05023     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
05024     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
05025     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<88>,
      ZPZV<26>, ZPZV<86>; };  // NOLINT
05026     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
05027     template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<86>; };  // NOLINT
05028     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<20>, ZPZV<86>; };  // NOLINT
05029     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
05030     template<> struct ConwayPolynomial<97, 1> { using ZPZ = aerobus::zpz<97>; using type =
      POLYV<ZPZV<1>, ZPZV<92>; };  // NOLINT
05031     template<> struct ConwayPolynomial<97, 2> { using ZPZ = aerobus::zpz<97>; using type =
      POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5>; };  // NOLINT
05032     template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; };  // NOLINT
05033     template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; };  // NOLINT
05034     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
05035     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
05036     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
05037     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
05038     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
05039     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
05040     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
05041     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
05042     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
05043     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
05044     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
05045     template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<99»; };  // NOLINT
05046     template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; };  // NOLINT
05047     template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99»; };  // NOLINT
05048     template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<78>, ZPZV<2»; };  // NOLINT
05049     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
05050     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
05051     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
05052     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
05053     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
05054     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
05055     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<31>, ZPZV<99»; };  // NOLINT
05056     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
05057     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<7>, ZPZV<99»; };  // NOLINT
05058     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<31>, ZPZV<99»; };  // NOLINT
05059     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<24>, ZPZV<99»; };  //
      NOLINT
05060     template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<98»; };  // NOLINT
05061     template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; };  // NOLINT
05062     template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; };  // NOLINT
05063     template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; };  // NOLINT
05064     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
05065     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
05066     template<> struct ConwayPolynomial<103, 7> { using ZPZ = aerobus::zpz<103>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; };  // NOLINT
05067     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
05068     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
05069     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
05070     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<5>, ZPZV<98»; };  // NOLINT
05071     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
05072     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<5>, ZPZV<98»; };  // NOLINT
05073     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<102>, ZPZV<8>, ZPZV<98»; };  // NOLINT
05074     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<2>, ZPZV<98»; };  //
      NOLINT
05075     template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<105»; };  // NOLINT
05076     template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; };  // NOLINT
05077     template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; };  // NOLINT
05078     template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; };  // NOLINT
05079     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
05080     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
05081     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
05082     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
05083     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
05084     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
05085     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<8>, ZPZV<105»; };  // NOLINT
05086     template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<37>, ZPZV<48>, ZPZV<6>, ZPZV<0>, ZPZV<61>,
      ZPZV<42>, ZPZV<57>, ZPZV<2»; };  // NOLINT
05087     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<4>, ZPZV<105»; };  // NOLINT
05088     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
05089     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<24>, ZPZV<105»; };  //
      NOLINT
05090     template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<103»; };  // NOLINT
05091     template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; };  // NOLINT
05092     template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };  // NOLINT
05093     template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; };  // NOLINT
05094     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
05095     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
05096     template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<103»; };  // NOLINT
05097     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
05098     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
05099     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
05100     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
05101     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
05102     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
05103     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<1>, ZPZV<103>>; };  // NOLINT
05104     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
05105     template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<110>>; };  // NOLINT
05106     template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3>>; };  // NOLINT
05107     template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<110>>; };  // NOLINT
05108     template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<3>>; };  // NOLINT
05109     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
05110     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
05111     template<> struct ConwayPolynomial<113, 7> { using ZPZ = aerobus::zpz<113>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<110>>; };  // NOLINT
05112     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
05113     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
05114     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
05115     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<3>, ZPZV<110>>; };  // NOLINT
05116     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
05117     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<4>, ZPZV<110>>; };  // NOLINT
05118     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<4>, ZPZV<110>>; };  // NOLINT
05119     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<22>, ZPZV<110>>; };  //
      NOLINT
05120     template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<124>>; };  // NOLINT
05121     template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3>>; };  // NOLINT
05122     template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124>>; };  // NOLINT
05123     template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<97>, ZPZV<3>>; };  // NOLINT
05124     template<> struct ConwayPolynomial<127, 5> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<124>>; };  // NOLINT
05125     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
05126     template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<124>>; };  // NOLINT
05127     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
05128     template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124>>;
      };  // NOLINT
05129     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
05130     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
05131     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
05132     template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<124>>; };  // NOLINT
05133     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
05134     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
05135     template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
      POLYV<ZPZV<1>, ZPZV<129»; };  // NOLINT
05136     template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
      POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; };  // NOLINT
05137     template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129»; };  // NOLINT
05138     template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; };  // NOLINT
05139     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
05140     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
05141     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
05142     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
05143     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
05144     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
05145     template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<6>, ZPZV<129»; };  // NOLINT
05146     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
05147     template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; };  // NOLINT
05148     template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<129»; };  // NOLINT
05149     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
05150     template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
      POLYV<ZPZV<1>, ZPZV<134»; };  // NOLINT
05151     template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
      POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; };  // NOLINT
05152     template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; };  // NOLINT
05153     template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3»; };  // NOLINT
05154     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
05155     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
05156     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
05157     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
05158     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
05159     template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
      ZPZV<3»; };  // NOLINT
05160     template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<1>, ZPZV<134»; };  // NOLINT
05161     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
05162     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
05163     template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134»; };  // NOLINT
05164     template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<134»; };  //
      NOLINT
05165     template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
      POLYV<ZPZV<1>, ZPZV<137»; };  // NOLINT
05166     template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2>>; };  // NOLINT
05167     template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137>>; };  // NOLINT
05168     template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2>>; };  // NOLINT
05169     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
05170     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
05171     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
05172     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
05173     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
05174     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
05175     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
05176     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
05177     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
05178     template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>>; };  // NOLINT
05179     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
05180     template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<147>>; };  // NOLINT
05181     template<> struct ConwayPolynomial<149, 2> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2>>; };  // NOLINT
05182     template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147>>; };  // NOLINT
05183     template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<107>, ZPZV<2>>; };  // NOLINT
05184     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
05185     template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<105>, ZPZV<33>, ZPZV<55>, ZPZV<2>>; };  // NOLINT
05186     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
05187     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
05188     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
05189     template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<74>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>,
      ZPZV<2>>; };  // NOLINT
05190     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
05191     template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<52>, ZPZV<9>,
      ZPZV<104>, ZPZV<110>, ZPZV<2>>; };  // NOLINT
05192     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
05193     template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<147>>; };  // NOLINT
05194     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
05195     template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<145>>; };  // NOLINT
05196     template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6>>; };  // NOLINT
05197     template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145>>; };  // NOLINT
05198     template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6>>; };  // NOLINT
05199     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
05200     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
```

```
05201     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
05202     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
05203     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
05204     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
05205     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
05206     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
05207     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
05208     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
05209     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
05210     template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<152»; };  // NOLINT
05211     template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»; };  // NOLINT
05212     template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<152»; };  // NOLINT
05213     template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5»; };  // NOLINT
05214     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
05215     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
05216     template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; };  // NOLINT
05217     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
05218     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
05219     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
05220     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
05221     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
05222     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
05223     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
05224     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
05225     template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<161»; };  // NOLINT
05226     template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; };  // NOLINT
05227     template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; };  // NOLINT
05228     template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2»; };  // NOLINT
05229     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
05230     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
05231     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
05232     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
05233     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
05234     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
05235     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<11>, ZPZV<161»; };  // NOLINT
05236     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
05237     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<24>, ZPZV<161»; };  // NOLINT
05238     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<71>, ZPZV<161»; };  // NOLINT
05239     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
05240     template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
      POLYV<ZPZV<1>, ZPZV<162»; };  // NOLINT
05241     template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
      POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; };  // NOLINT
05242     template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; };  // NOLINT
05243     template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; };  // NOLINT
05244     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
05245     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
05246     template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<162»; };  // NOLINT
05247     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
05248     template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<122>, ZPZV<162»;
      };  // NOLINT
05249     template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
      ZPZV<148>, ZPZV<5»; };  // NOLINT
05250     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<24>, ZPZV<162»; };  // NOLINT
05251     template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<10>, ZPZV<142>, ZPZV<131>,
      ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5»; };  // NOLINT
05252     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<10>, ZPZV<162»; };  // NOLINT
05253     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<32>, ZPZV<162»; };  // NOLINT
05254     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<14>, ZPZV<162»; };  //
      NOLINT
05255     template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
      POLYV<ZPZV<1>, ZPZV<171»; };  // NOLINT
05256     template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
      POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; };  // NOLINT
05257     template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<171»; };  // NOLINT
05258     template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; };  // NOLINT
05259     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
05260     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
05261     template<> struct ConwayPolynomial<173, 7> { using ZPZ = aerobus::zpz<173>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<171»; };  // NOLINT
05262     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
05263     template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
      };  // NOLINT
05264     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
05265     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<12>, ZPZV<171»; };  // NOLINT
05266     template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<64>, ZPZV<46>, ZPZV<166>, ZPZV<0>,
      ZPZV<159>, ZPZV<22>, ZPZV<2»; };  // NOLINT
05267     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
05268     template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<171»; };  // NOLINT
05269     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
05270     template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
      POLYV<ZPZV<1>, ZPZV<177»; };  // NOLINT
05271     template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
      POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; };  // NOLINT
05272     template<> struct ConwayPolynomial<179, 3> { using ZPZ = aerobus::zpz<179>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<177»; };  // NOLINT
05273     template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<2»; };  // NOLINT
05274     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
05275     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
05276     template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<177»; };  // NOLINT
05277     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
05278     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
05279     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
05280     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
05281     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
05282     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
05283     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
05284     template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<177»; };  //
      NOLINT
05285     template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
      POLYV<ZPZV<1>, ZPZV<179»; };  // NOLINT
05286     template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
      POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; };  // NOLINT
05287     template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<179»; };  // NOLINT
05288     template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; };  // NOLINT
05289     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
05290     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
05291     template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<179»; };  // NOLINT
05292     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
05293     template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
      };  // NOLINT
05294     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
05295     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
05296     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
05297     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
05298     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
05299     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
```

```
05300     template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
      POLYV<ZPZV<1>, ZPZV<172>>; };  // NOLINT
05301     template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
      POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19>>; };  // NOLINT
05302     template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172>>; };  // NOLINT
05303     template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19>>; };  // NOLINT
05304     template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172>>; };  // NOLINT
05305     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
05306     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
05307     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
05308     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
05309     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
05310     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
05311     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
05312     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
05313     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
05314     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<0>, ZPZV<190>, ZPZV<2>, ZPZV<172>>; };  //
      NOLINT
05315     template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
      POLYV<ZPZV<1>, ZPZV<188>>; };  // NOLINT
05316     template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
      POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5>>; };  // NOLINT
05317     template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188>>; };  // NOLINT
05318     template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<148>, ZPZV<5>>; };  // NOLINT
05319     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
05320     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
05321     template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<188>>; };  // NOLINT
05322     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
05323     template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<168>, ZPZV<27>, ZPZV<188>>;
      };  // NOLINT
05324     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
05325     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
05326     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
05327     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
05328     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
05329     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
05330     template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
      POLYV<ZPZV<1>, ZPZV<195>>; };  // NOLINT
05331     template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
      POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2>>; };  // NOLINT
05332     template<> struct ConwayPolynomial<197, 3> { using ZPZ = aerobus::zpz<197>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<195>>; };  // NOLINT
05333     template<> struct ConwayPolynomial<197, 4> { using ZPZ = aerobus::zpz<197>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<124>, ZPZV<2>>; };  // NOLINT
05334     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
05335     template<> struct ConwayPolynomial<197, 6> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<124>, ZPZV<79>, ZPZV<173>, ZPZV<2»; };  // NOLINT
05336     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
05337     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
05338     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
05339     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
05340     template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<14>, ZPZV<195»; };  // NOLINT
05341     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
05342     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<39>, ZPZV<195»; };  // NOLINT
05343     template<> struct ConwayPolynomial<197, 17> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<195»; };  // NOLINT
05344     template<> struct ConwayPolynomial<197, 19> { using ZPZ = aerobus::zpz<197>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<195»; };  //
    NOLINT
05345     template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<196»; };  // NOLINT
05346     template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; };  // NOLINT
05347     template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; };  // NOLINT
05348     template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; };  // NOLINT
05349     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
05350     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
05351     template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; };  // NOLINT
05352     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
05353     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
05354     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
05355     template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<1>, ZPZV<196»; };  // NOLINT
05356     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
05357     template<> struct ConwayPolynomial<199, 13> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<196»; };  // NOLINT
05358     template<> struct ConwayPolynomial<199, 17> { using ZPZ = aerobus::zpz<199>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
    ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<196»; };  // NOLINT
05359     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
05360     template<> struct ConwayPolynomial<211, 1> { using ZPZ = aerobus::zpz<211>; using type =
    POLYV<ZPZV<1>, ZPZV<209»; };  // NOLINT
05361     template<> struct ConwayPolynomial<211, 2> { using ZPZ = aerobus::zpz<211>; using type =
    POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; };  // NOLINT
05362     template<> struct ConwayPolynomial<211, 3> { using ZPZ = aerobus::zpz<211>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; };  // NOLINT
05363     template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<161>, ZPZV<2»; };  // NOLINT
05364     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
05365     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
05366     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
05367     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
05368     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
05369     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
05370     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
05371     template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>,
      ZPZV<84>, ZPZV<27>, ZPZV<2>>; };  // NOLINT
05372     template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209>>; };  // NOLINT
05373     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<12>, ZPZV<209>>; };  // NOLINT
05374     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
05375     template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<220>>; };  // NOLINT
05376     template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3>>; };  // NOLINT
05377     template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220>>; };  // NOLINT
05378     template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3>>; };  // NOLINT
05379     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
05380     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
05381     template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<220>>; };  // NOLINT
05382     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
05383     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
05384     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
05385     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
05386     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
05387     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
05388     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
05389     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
05390     template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<225>>; };  // NOLINT
05391     template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2>>; };  // NOLINT
05392     template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225>>; };  // NOLINT
05393     template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2>>; };  // NOLINT
05394     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
05395     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
05396     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
05397     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
05398     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
05399     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
05400     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
05401     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
05402     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
05403     template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<225>>; };  // NOLINT
05404     template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<225>>; };  //
      NOLINT
05405     template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<223>>; };  // NOLINT
05406     template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6>>; };  // NOLINT
05407     template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223>>; };  // NOLINT
05408     template<> struct ConwayPolynomial<229, 4> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6>>; };  // NOLINT
05409     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
05410     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
05411     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
05412     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
05413     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
05414     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
05415     template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<2>, ZPZV<223>>; };  // NOLINT
05416     template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<6>, ZPZV<172>,
      ZPZV<9>, ZPZV<145>, ZPZV<6>>; };  // NOLINT
05417     template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223>>; };  // NOLINT
05418     template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<223>>; };  // NOLINT
05419     template<> struct ConwayPolynomial<229, 19> { using ZPZ = aerobus::zpz<229>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<228>, ZPZV<15>, ZPZV<223>>; };  //
      NOLINT
05420     template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<230>>; };  // NOLINT
05421     template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3>>; };  // NOLINT
05422     template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230>>; };  // NOLINT
05423     template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3>>; };  // NOLINT
05424     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
05425     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
05426     template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230>>; };  // NOLINT
05427     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
05428     template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<56>, ZPZV<146>, ZPZV<230>>;
      };  // NOLINT
05429     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
05430     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
05431     template<> struct ConwayPolynomial<233, 12> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, 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
05432     template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230>>; };  // NOLINT
05433     template<> struct ConwayPolynomial<233, 17> { using ZPZ = aerobus::zpz<233>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230>>; };  // NOLINT
05434     template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<230»; };  //
      NOLINT
05435     template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
      POLYV<ZPZV<1>, ZPZV<232»; };  // NOLINT
05436     template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
      POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; };  // NOLINT
05437     template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232»; };  // NOLINT
05438     template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; };  // NOLINT
05439     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
05440     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
05441     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
05442     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
05443     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
05444     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
05445     template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<8>, ZPZV<232»; };  // NOLINT
05446     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
05447     template<> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; };  // NOLINT
05448     template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<232»; };  // NOLINT
05449     template<> struct ConwayPolynomial<239, 19> { using ZPZ = aerobus::zpz<239>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<232»; };  //
      NOLINT
05450     template<> struct ConwayPolynomial<241, 1> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<234»; };  // NOLINT
05451     template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; };  // NOLINT
05452     template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; };  // NOLINT
05453     template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<152>, ZPZV<7»; };  // NOLINT
05454     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
05455     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
05456     template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234»; };  // NOLINT
05457     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
05458     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
05459     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
05460     template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<3>, ZPZV<234»; };  // NOLINT
05461     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
05462     template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
      ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<234»; };  // NOLINT
05463     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<14>, ZPZV<234»; };  // NOLINT
05464     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
05465     template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<245»; };  // NOLINT
05466     template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; };  // NOLINT
05467     template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; };  // NOLINT
```

```
05468     template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; };  // NOLINT
05469     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
05470     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
05471     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
05472     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
05473     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
05474     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
05475     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<26>, ZPZV<245»; };  // NOLINT
05476     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
05477     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<15>, ZPZV<245»; };  // NOLINT
05478     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<9>, ZPZV<245»; };  // NOLINT
05479     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<8>, ZPZV<245»; };  //
      NOLINT
05480     template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<254»; };  // NOLINT
05481     template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; };  // NOLINT
05482     template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254»; };  // NOLINT
05483     template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<187>, ZPZV<3»; };  // NOLINT
05484     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
05485     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
05486     template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<254»; };  // NOLINT
05487     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
05488     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
05489     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
05490     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<40>, ZPZV<254»; };  // NOLINT
05491     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
05492     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<5>, ZPZV<254»; };  // NOLINT
05493     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<8>, ZPZV<254»; };  // NOLINT
05494     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<10>, ZPZV<254»; };  //
      NOLINT
05495     template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<258»; };  // NOLINT
05496     template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; };  // NOLINT
05497     template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; };  // NOLINT
05498     template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; };  // NOLINT
05499     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
05500     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
05501     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
05502     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
05503     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
05504     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
05505     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<2>, ZPZV<258»; };  // NOLINT
05506     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
05507     template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<267»; };  // NOLINT
05508     template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; };  // NOLINT
05509     template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<267»; };  // NOLINT
05510     template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<262>, ZPZV<2»; };  // NOLINT
05511     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
05512     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
05513     template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<267»; };  // NOLINT
05514     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
05515     template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<214>, ZPZV<267>, ZPZV<267»;
      };  // NOLINT
05516     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
05517     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<20>, ZPZV<267»; };  // NOLINT
05518     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
05519     template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<265»; };  // NOLINT
05520     template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; };  // NOLINT
05521     template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; };  // NOLINT
05522     template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; };  // NOLINT
05523     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
05524     template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<6»; };  // NOLINT
05525     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
05526     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
05527     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
05528     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
05529     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<10>, ZPZV<265»; };  // NOLINT
05530     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
05531     template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<272»; };  // NOLINT
05532     template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; };  // NOLINT
05533     template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; };  // NOLINT
05534     template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5»; };  // NOLINT
05535     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
05536     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
05537     template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<272»; };  // NOLINT
05538     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
05539     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
05540     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
05541     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<5>, ZPZV<272»; };  // NOLINT
05542     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
05543     template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<278»; };  // NOLINT
05544     template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; };  // NOLINT
05545     template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; };  // NOLINT
05546     template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<176>, ZPZV<3»; };  // NOLINT
05547     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
05548     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
05549     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
05550     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
05551     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
05552     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
05553     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<36>, ZPZV<278»; };  // NOLINT
05554     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
05555     template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<280»; };  // NOLINT
05556     template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; };  // NOLINT
05557     template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; };  // NOLINT
05558     template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; };  // NOLINT
05559     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
05560     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
05561     template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<280»; };  // NOLINT
05562     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
05563     template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<65>, ZPZV<280»;
      };  // NOLINT
05564     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
05565     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<4>, ZPZV<280»; };  // NOLINT
05566     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
05567     template<> struct ConwayPolynomial<293, 1> { using ZPZ = aerobus::zpz<293>; using type =
      POLYV<ZPZV<1>, ZPZV<291»; };  // NOLINT
05568     template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
      POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; };  // NOLINT
05569     template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; };  // NOLINT
05570     template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; };  // NOLINT
05571     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
05572     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
05573     template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<291»; };  // NOLINT
05574     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
05575     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
05576     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
05577     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<3>, ZPZV<291»; };  // NOLINT
05578     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
05579     template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
      POLYV<ZPZV<1>, ZPZV<302»; };  // NOLINT
05580     template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
      POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; };  // NOLINT
05581     template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<302»; };  // NOLINT
05582     template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<239>, ZPZV<5»; };  // NOLINT
05583     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
05584     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
05585     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
05586     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
05587     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
05588     template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
      POLYV<ZPZV<1>, ZPZV<294»; };  // NOLINT
05589     template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
      POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; };  // NOLINT
05590     template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; };  // NOLINT
05591     template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; };  // NOLINT
05592     template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; };  // NOLINT
05593     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
05594     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
05595     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
05596     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
05597     template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
      POLYV<ZPZV<1>, ZPZV<303»; };  // NOLINT
05598     template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
      POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; };  // NOLINT
05599     template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; };  // NOLINT
05600     template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<239>, ZPZV<10»; };  // NOLINT
05601     template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; };  // NOLINT
05602     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
05603     template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<303»; };  // NOLINT
05604     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
05605     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
05606     template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<315»; };  // NOLINT
05607     template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; };  // NOLINT
05608     template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; };  // NOLINT
05609     template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; };  // NOLINT
05610     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
05611     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
05612     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
05613     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
05614     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
05615     template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<328»; };  // NOLINT
05616     template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3»; };  // NOLINT
05617     template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<328»; };  // NOLINT
05618     template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; };  // NOLINT
05619     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
05620     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
05621     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
05622     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
05623     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<194>, ZPZV<210>, ZPZV<328»;
      };  // NOLINT
05624     template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<327»; };  // NOLINT
05625     template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; };  // NOLINT
05626     template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; };  // NOLINT
05627     template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; };  // NOLINT
05628     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
05629     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
05630     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
05631     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
05632     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
05633     template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<345»; };  // NOLINT
05634     template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; };  // NOLINT
05635     template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; };  // NOLINT
05636     template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; };  // NOLINT
05637     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
05638     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
05639     template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<345»; };  // NOLINT
05640     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
05641     template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<235>, ZPZV<252>, ZPZV<345»;
      };  // NOLINT
05642     template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<347»; };  // NOLINT
05643     template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; };  // NOLINT
05644     template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; };  // NOLINT
05645     template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2»; };  // NOLINT
05646     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
05647     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
05648     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
05649     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
05650     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
```

```
05651     template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<350>>; };  // NOLINT
05652     template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3>>; };  // NOLINT
05653     template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350>>; };  // NOLINT
05654     template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3>>; };  // NOLINT
05655     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
05656     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
05657     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
05658     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
05659     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
05660     template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<352>>; };  // NOLINT
05661     template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7>>; };  // NOLINT
05662     template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352>>; };  // NOLINT
05663     template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<229>, ZPZV<7>>; };  // NOLINT
05664     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
05665     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
05666     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
05667     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
05668     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
05669     template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<361>>; };  // NOLINT
05670     template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6>>; };  // NOLINT
05671     template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361>>; };  // NOLINT
05672     template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6>>; };  // NOLINT
05673     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
05674     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
05675     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
05676     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
05677     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
05678     template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<371>>; };  // NOLINT
05679     template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2>>; };  // NOLINT
05680     template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371>>; };  // NOLINT
05681     template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2>>; };  // NOLINT
05682     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
05683     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
05684     template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<371>>; };  // NOLINT
05685     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
05686     template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<370>, ZPZV<371>>;
      };  // NOLINT
05687     template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<377>>; };  // NOLINT
05688     template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2>>; };  // NOLINT
05689     template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377>>; };  // NOLINT
05690     template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2»; };  // NOLINT
05691     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
05692     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
05693     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
05694     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
05695     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
05696     template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<378»; };  // NOLINT
05697     template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; };  // NOLINT
05698     template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; };  // NOLINT
05699     template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; };  // NOLINT
05700     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
05701     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
05702     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
05703     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
05704     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
05705     template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<387»; };  // NOLINT
05706     template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; };  // NOLINT
05707     template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; };  // NOLINT
05708     template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<266>, ZPZV<2»; };  // NOLINT
05709     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
05710     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
05711     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
05712     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
05713     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
05714     template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<392»; };  // NOLINT
05715     template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; };  // NOLINT
05716     template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; };  // NOLINT
05717     template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; };  // NOLINT
05718     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
05719     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
05720     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
05721     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
05722     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
05723     template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<398»; };  // NOLINT
05724     template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; };  // NOLINT
05725     template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; };  // NOLINT
05726     template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; };  // NOLINT
05727     template<> struct ConwayPolynomial<401, 5> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<398»; };  // NOLINT
05728     template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<81>, ZPZV<51>, ZPZV<3»; };  // NOLINT
05729     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
```

```
05730     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
05731     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
05732     template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<388>>; };  // NOLINT
05733     template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21>>; };  // NOLINT
05734     template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388>>; };  // NOLINT
05735     template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21>>; };  // NOLINT
05736     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
05737     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
05738     template<> struct ConwayPolynomial<409, 7> { using ZPZ = aerobus::zpz<409>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<388>>; };  // NOLINT
05739     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
05740     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
05741     template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<417>>; };  // NOLINT
05742     template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2>>; };  // NOLINT
05743     template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417>>; };  // NOLINT
05744     template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2>>; };  // NOLINT
05745     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
05746     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
05747     template<> struct ConwayPolynomial<419, 7> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417>>; };  // NOLINT
05748     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
05749     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<93>, ZPZV<386>, ZPZV<417>>;
      };  // NOLINT
05750     template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<419>>; };  // NOLINT
05751     template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2>>; };  // NOLINT
05752     template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<419>>; };  // NOLINT
05753     template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2>>; };  // NOLINT
05754     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
05755     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
05756     template<> struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<419>>; };  // NOLINT
05757     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
05758     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
05759     template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<424>>; };  // NOLINT
05760     template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7>>; };  // NOLINT
05761     template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424>>; };  // NOLINT
05762     template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<323>, ZPZV<7>>; };  // NOLINT
05763     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
05764     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
05765     template<> struct ConwayPolynomial<431, 7> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424>>; };  // NOLINT
05766     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
05767     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
05768     template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<428»; };  // NOLINT
05769     template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; };  // NOLINT
05770     template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; };  // NOLINT
05771     template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; };  // NOLINT
05772     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
05773     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
05774     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
05775     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
05776     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
05777     template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<424»; };  // NOLINT
05778     template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; };  // NOLINT
05779     template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; };  // NOLINT
05780     template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; };  // NOLINT
05781     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
05782     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
05783     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
05784     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
05785     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
05786     template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<441»; };  // NOLINT
05787     template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; };  // NOLINT
05788     template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; };  // NOLINT
05789     template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; };  // NOLINT
05790     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
05791     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
05792     template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<441»; };  // NOLINT
05793     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
05794     template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109>, ZPZV<441»;
      };  // NOLINT
05795     template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<446»; };  // NOLINT
05796     template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; };  // NOLINT
05797     template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; };  // NOLINT
05798     template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; };  // NOLINT
05799     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
05800     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
05801     template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446»; };  // NOLINT
05802     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
05803     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
05804     template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<444»; };  // NOLINT
05805     template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; };  // NOLINT
05806     template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444»; };  // NOLINT
05807     template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; };  // NOLINT
```

```
05808     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
05809     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
05810     template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<444»; };  // NOLINT
05811     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
05812     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
05813     template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<459»; };  // NOLINT
05814     template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; };  // NOLINT
05815     template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; };  // NOLINT
05816     template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; };  // NOLINT
05817     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
05818     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
05819     template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<459»; };  // NOLINT
05820     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
05821     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
05822     template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<460»; };  // NOLINT
05823     template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; };  // NOLINT
05824     template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<460»; };  // NOLINT
05825     template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; };  // NOLINT
05826     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
05827     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
05828     template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<460»; };  // NOLINT
05829     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
05830     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
05831     template<> struct ConwayPolynomial<467, 1> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<465»; };  // NOLINT
05832     template<> struct ConwayPolynomial<467, 2> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; };  // NOLINT
05833     template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; };  // NOLINT
05834     template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; };  // NOLINT
05835     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
05836     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
05837     template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465»; };  // NOLINT
05838     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
05839     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
05840     template<> struct ConwayPolynomial<479, 1> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<466»; };  // NOLINT
05841     template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; };  // NOLINT
05842     template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; };  // NOLINT
05843     template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; };  // NOLINT
05844     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
05845     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
05846     template<> struct ConwayPolynomial<479, 7> { using ZPZ = aerobus::zpz<479>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<466»; };  // NOLINT
05847     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
05848     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
05849     template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<484»; };  // NOLINT
05850     template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; };  // NOLINT
05851     template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; };  // NOLINT
05852     template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; };  // NOLINT
05853     template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484»; };  // NOLINT
05854     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
05855     template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<484»; };  // NOLINT
05856     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
05857     template<> struct ConwayPolynomial<487, 9> { using ZPZ = aerobus::zpz<487>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<271>, ZPZV<447>, ZPZV<484»;
      };  // NOLINT
05858     template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<489»; };  // NOLINT
05859     template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; };  // NOLINT
05860     template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; };  // NOLINT
05861     template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2»; };  // NOLINT
05862     template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; };  // NOLINT
05863     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
05864     template<> struct ConwayPolynomial<491, 7> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; };  // NOLINT
05865     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
05866     template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<453>, ZPZV<489»;
      };  // NOLINT
05867     template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<492»; };  // NOLINT
05868     template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; };  // NOLINT
05869     template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; };  // NOLINT
05870     template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; };  // NOLINT
05871     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
05872     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
05873     template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<492»; };  // NOLINT
05874     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
05875     template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<491>, ZPZV<222>, ZPZV<492»;
      };  // NOLINT
05876     template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<498»; };  // NOLINT
05877     template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; };  // NOLINT
05878     template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; };  // NOLINT
05879     template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; };  // NOLINT
05880     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
05881     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
05882     template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<498»; };  // NOLINT
05883     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
05884     template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498»;
      };  // NOLINT
05885     template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<507»; };  // NOLINT
```

```
05886     template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2>>; };  // NOLINT
05887     template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507>>; };  // NOLINT
05888     template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2>>; };  // NOLINT
05889     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
05890     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
05891     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
05892     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
05893     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
05894     template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<518>>; };  // NOLINT
05895     template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3>>; };  // NOLINT
05896     template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518>>; };  // NOLINT
05897     template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<509>, ZPZV<3>>; };  // NOLINT
05898     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
05899     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
05900     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
05901     template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<3>>; };  //
      NOLINT
05902     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
05903     template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<521>>; };  // NOLINT
05904     template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2>>; };  // NOLINT
05905     template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<521>>; };  // NOLINT
05906     template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2>>; };  // NOLINT
05907     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
05908     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
05909     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
05910     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
05911     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
05912     template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<539>>; };  // NOLINT
05913     template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2>>; };  // NOLINT
05914     template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539>>; };  // NOLINT
05915     template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2>>; };  // NOLINT
05916     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
05917     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
05918     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
05919     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
05920     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
05921     template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<545>>; };  // NOLINT
05922     template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2>>; };  // NOLINT
05923     template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545>>; };  // NOLINT
05924     template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2>>; };  // NOLINT
05925     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
05926     template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; };  // NOLINT
05927     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
05928     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
05929     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
05930     template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
    POLYV<ZPZV<1>, ZPZV<555»; };  // NOLINT
05931     template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
    POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; };  // NOLINT
05932     template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555»; };  // NOLINT
05933     template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; };  // NOLINT
05934     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
05935     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
05936     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
05937     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
05938     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
05939     template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
    POLYV<ZPZV<1>, ZPZV<561»; };  // NOLINT
05940     template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
    POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; };  // NOLINT
05941     template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; };  // NOLINT
05942     template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; };  // NOLINT
05943     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
05944     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
05945     template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<561»; };  // NOLINT
05946     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
05947     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
05948     template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
    POLYV<ZPZV<1>, ZPZV<566»; };  // NOLINT
05949     template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
    POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; };  // NOLINT
05950     template<> struct ConwayPolynomial<569, 3> { using ZPZ = aerobus::zpz<569>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; };  // NOLINT
05951     template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; };  // NOLINT
05952     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
05953     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
05954     template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<566»; };  // NOLINT
05955     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
05956     template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<478>, ZPZV<566>, ZPZV<566»;
    };  // NOLINT
05957     template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
    POLYV<ZPZV<1>, ZPZV<568»; };  // NOLINT
05958     template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
    POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; };  // NOLINT
05959     template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; };  // NOLINT
05960     template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; };  // NOLINT
05961     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
05962     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
05963     template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<568»; };  // NOLINT
05964     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
05965     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
05966     template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572>; };  // NOLINT
05967     template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5>; };  // NOLINT
05968     template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572>; };  // NOLINT
05969     template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<494>, ZPZV<5>; };  // NOLINT
05970     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
05971     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
05972     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
05973     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
05974     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
05975     template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<585>; };  // NOLINT
05976     template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2>; };  // NOLINT
05977     template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585>; };  // NOLINT
05978     template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2>; };  // NOLINT
05979     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
05980     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
05981     template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<585>; };  // NOLINT
05982     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
05983     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
05984     template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<590>; };  // NOLINT
05985     template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3>; };  // NOLINT
05986     template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590>; };  // NOLINT
05987     template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3>; };  // NOLINT
05988     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
05989     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
05990     template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<590>; };  // NOLINT
05991     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
05992     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
05993     template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<592>; };  // NOLINT
05994     template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7>; };  // NOLINT
05995     template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592>; };  // NOLINT
05996     template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7>; };  // NOLINT
05997     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
05998     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
05999     template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<592>; };  // NOLINT
06000     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
06001     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
06002     template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<594>; };  // NOLINT
06003     template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; };  // NOLINT
06004     template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; };  // NOLINT
06005     template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; };  // NOLINT
06006     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
06007     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
06008     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
06009     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
06010     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
06011     template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<604»; };  // NOLINT
06012     template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; };  // NOLINT
06013     template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; };  // NOLINT
06014     template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; };  // NOLINT
06015     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
06016     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
06017     template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<604»; };  // NOLINT
06018     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
06019     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
06020     template<> struct ConwayPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<611»; };  // NOLINT
06021     template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; };  // NOLINT
06022     template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; };  // NOLINT
06023     template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; };  // NOLINT
06024     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
06025     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
06026     template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<611»; };  // NOLINT
06027     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
06028     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
06029     template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<614»; };  // NOLINT
06030     template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; };  // NOLINT
06031     template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; };  // NOLINT
06032     template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; };  // NOLINT
06033     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
06034     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
06035     template<> struct ConwayPolynomial<617, 7> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<614»; };  // NOLINT
06036     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
06037     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
06038     template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<617»; };  // NOLINT
06039     template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; };  // NOLINT
06040     template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; };  // NOLINT
06041     template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<492>, ZPZV<2»; };  // NOLINT
06042     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
```

```
06043     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
06044     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
06045     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
06046     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<579>, ZPZV<310>, ZPZV<617>>;
      };  // NOLINT
06047     template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<628>>; };  // NOLINT
06048     template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3>>; };  // NOLINT
06049     template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628>>; };  // NOLINT
06050     template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<3>>; };  // NOLINT
06051     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
06052     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
06053     template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628>>; };  // NOLINT
06054     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
06055     template<> struct ConwayPolynomial<631, 9> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<296>, ZPZV<413>, ZPZV<628>>;
      };  // NOLINT
06056     template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<638>>; };  // NOLINT
06057     template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3>>; };  // NOLINT
06058     template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638>>; };  // NOLINT
06059     template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<3>>; };  // NOLINT
06060     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
06061     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
06062     template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638>>; };  // NOLINT
06063     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
06064     template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<141>, ZPZV<638>>;
      };  // NOLINT
06065     template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<632>>; };  // NOLINT
06066     template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11>>; };  // NOLINT
06067     template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632>>; };  // NOLINT
06068     template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<11>>; };  // NOLINT
06069     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
06070     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
06071     template<> struct ConwayPolynomial<643, 7> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<632>>; };  // NOLINT
06072     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
06073     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
06074     template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<642>>; };  // NOLINT
06075     template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5>>; };  // NOLINT
06076     template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642>>; };  // NOLINT
06077     template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5>>; };  // NOLINT
06078     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
06079     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
06080     template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<642>>; };  // NOLINT
06081     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
```

```
06082     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
06083     template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<651>; };  // NOLINT
06084     template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2>; };  // NOLINT
06085     template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651>; };  // NOLINT
06086     template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2>; };  // NOLINT
06087     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
06088     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
06089     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
06090     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
06091     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<365>, ZPZV<60>, ZPZV<651>;
      };  // NOLINT
06092     template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<657>; };  // NOLINT
06093     template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2>; };  // NOLINT
06094     template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657>; };  // NOLINT
06095     template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2>; };  // NOLINT
06096     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
06097     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
06098     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
06099     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
06100     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
06101     template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<659>; };  // NOLINT
06102     template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2>; };  // NOLINT
06103     template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659>; };  // NOLINT
06104     template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2>; };  // NOLINT
06105     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
06106     template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2>; };  // NOLINT
06107     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
06108     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
06109     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
06110     template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<668>; };  // NOLINT
06111     template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5>; };  // NOLINT
06112     template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668>; };  // NOLINT
06113     template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5>; };  // NOLINT
06114     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
06115     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
06116     template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<668>; };  // NOLINT
06117     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
06118     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
06119     template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<675>; };  // NOLINT
06120     template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2>; };  // NOLINT
```

```
06121     template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; };  // NOLINT
06122     template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; };  // NOLINT
06123     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
06124     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
06125     template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<675»; };  // NOLINT
06126     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
06127     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
06128     template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<678»; };  // NOLINT
06129     template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; };  // NOLINT
06130     template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; };  // NOLINT
06131     template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; };  // NOLINT
06132     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
06133     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
06134     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
06135     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
06136     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
06137     template<> struct ConwayPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<688»; };  // NOLINT
06138     template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; };  // NOLINT
06139     template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; };  // NOLINT
06140     template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; };  // NOLINT
06141     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
06142     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
06143     template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; };  // NOLINT
06144     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
06145     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
06146     template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<699»; };  // NOLINT
06147     template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; };  // NOLINT
06148     template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; };  // NOLINT
06149     template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; };  // NOLINT
06150     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
06151     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
06152     template<> struct ConwayPolynomial<701, 7> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; };  // NOLINT
06153     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
06154     template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<459>, ZPZV<373>, ZPZV<699»;
      };  // NOLINT
06155     template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<707»; };  // NOLINT
06156     template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; };  // NOLINT
06157     template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; };  // NOLINT
06158     template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; };  // NOLINT
06159     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
06160     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
06161     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
06162     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
06163     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
06164     template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<708»; };  // NOLINT
06165     template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; };  // NOLINT
06166     template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; };  // NOLINT
06167     template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<602>, ZPZV<11»; };  // NOLINT
06168     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
06169     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
06170     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
06171     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
06172     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
06173     template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<722»; };  // NOLINT
06174     template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; };  // NOLINT
06175     template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<722»; };  // NOLINT
06176     template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5»; };  // NOLINT
06177     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
06178     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
06179     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
06180     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
06181     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
06182     template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<727»; };  // NOLINT
06183     template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; };  // NOLINT
06184     template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; };  // NOLINT
06185     template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539>, ZPZV<6»; };  // NOLINT
06186     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
06187     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
06188     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
06189     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
06190     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
06191     template<> struct ConwayPolynomial<739, 1> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<736»; };  // NOLINT
06192     template<> struct ConwayPolynomial<739, 2> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<734>, ZPZV<3»; };  // NOLINT
06193     template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; };  // NOLINT
06194     template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<3»; };  // NOLINT
06195     template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; };  // NOLINT
06196     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
06197     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
06198     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
06199     template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<616>, ZPZV<81>, ZPZV<736»;
      };  // NOLINT
06200     template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<738»; };  // NOLINT
06201     template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; };  // NOLINT
06202     template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738»; };  // NOLINT
06203     template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; };  // NOLINT
06204     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
06205     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
06206     template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; };  // NOLINT
06207     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
06208     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
06209     template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<748»; };  // NOLINT
06210     template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; };  // NOLINT
06211     template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; };  // NOLINT
06212     template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; };  // NOLINT
06213     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
06214     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
06215     template<> struct ConwayPolynomial<751, 7> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; };  // NOLINT
06216     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
06217     template<> struct ConwayPolynomial<751, 9> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<703>, ZPZV<489>, ZPZV<748»;
      };  // NOLINT
06218     template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<755»; };  // NOLINT
06219     template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; };  // NOLINT
06220     template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; };  // NOLINT
06221     template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537>, ZPZV<2»; };  // NOLINT
06222     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
06223     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
06224     template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<755»; };  // NOLINT
06225     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
06226     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
06227     template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<755»; };  // NOLINT
06228     template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; };  // NOLINT
06229     template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; };  // NOLINT
06230     template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6»; };  // NOLINT
06231     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
06232     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
06233     template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<755»; };  // NOLINT
06234     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
06235     template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
      };  // NOLINT
06236     template<> struct ConwayPolynomial<769, 1> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<758»; };  // NOLINT
06237     template<> struct ConwayPolynomial<769, 2> { using ZPZ = aerobus::zpz<769>; using type =
      POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; };  // NOLINT
06238     template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; };  // NOLINT
06239     template<> struct ConwayPolynomial<769, 4> { using ZPZ = aerobus::zpz<769>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; };  // NOLINT
06240     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
06241     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
06242     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
06243     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
06244     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
06245     template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
    POLYV<ZPZV<1>, ZPZV<771»; };  // NOLINT
06246     template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
    POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; };  // NOLINT
06247     template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<771»; };  // NOLINT
06248     template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; };  // NOLINT
06249     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
06250     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
06251     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
06252     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
06253     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
06254     template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
    POLYV<ZPZV<1>, ZPZV<785»; };  // NOLINT
06255     template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
    POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; };  // NOLINT
06256     template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; };  // NOLINT
06257     template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<605>, ZPZV<2»; };  // NOLINT
06258     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
06259     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
06260     template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<785»; };  // NOLINT
06261     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
06262     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
06263     template<> struct ConwayPolynomial<797, 1> { using ZPZ = aerobus::zpz<797>; using type =
    POLYV<ZPZV<1>, ZPZV<795»; };  // NOLINT
06264     template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
    POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; };  // NOLINT
06265     template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; };  // NOLINT
06266     template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2»; };  // NOLINT
06267     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
06268     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
06269     template<> struct ConwayPolynomial<797, 7> { using ZPZ = aerobus::zpz<797>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<795»; };  // NOLINT
06270     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
06271     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
06272     template<> struct ConwayPolynomial<809, 1> { using ZPZ = aerobus::zpz<809>; using type =
    POLYV<ZPZV<1>, ZPZV<806»; };  // NOLINT
06273     template<> struct ConwayPolynomial<809, 2> { using ZPZ = aerobus::zpz<809>; using type =
    POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; };  // NOLINT
06274     template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; };  // NOLINT
06275     template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; };  // NOLINT
06276     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
06277     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
```

```
06278     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
06279     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
06280     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
06281     template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<808»; };  // NOLINT
06282     template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; };  // NOLINT
06283     template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<808»; };  // NOLINT
06284     template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; };  // NOLINT
06285     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
06286     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
06287     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
06288     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
06289     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
06290     template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<819»; };  // NOLINT
06291     template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; };  // NOLINT
06292     template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; };  // NOLINT
06293     template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2»; };  // NOLINT
06294     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
06295     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
06296     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
06297     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
06298     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
06299     template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<820»; };  // NOLINT
06300     template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; };  // NOLINT
06301     template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; };  // NOLINT
06302     template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; };  // NOLINT
06303     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
06304     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
06305     template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<820»; };  // NOLINT
06306     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
06307     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
06308     template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<825»; };  // NOLINT
06309     template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; };  // NOLINT
06310     template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; };  // NOLINT
06311     template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; };  // NOLINT
06312     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
06313     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
06314     template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<825»; };  // NOLINT
06315     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
06316     template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<372>, ZPZV<825»;
```

```
      };  // NOLINT
06317     template<> struct ConwayPolynomial<829, 1> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<827»; };  // NOLINT
06318     template<> struct ConwayPolynomial<829, 2> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; };  // NOLINT
06319     template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; };  // NOLINT
06320     template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; };  // NOLINT
06321     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
06322     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
06323     template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<827»; };  // NOLINT
06324     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
06325     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
06326     template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<828»; };  // NOLINT
06327     template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; };  // NOLINT
06328     template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; };  // NOLINT
06329     template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; };  // NOLINT
06330     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
06331     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
06332     template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828»; };  // NOLINT
06333     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
06334     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
06335     template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
      POLYV<ZPZV<1>, ZPZV<851»; };  // NOLINT
06336     template<> struct ConwayPolynomial<853, 2> { using ZPZ = aerobus::zpz<853>; using type =
      POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; };  // NOLINT
06337     template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; };  // NOLINT
06338     template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; };  // NOLINT
06339     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
06340     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
06341     template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<851»; };  // NOLINT
06342     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
06343     template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<677>, ZPZV<821>, ZPZV<851»;
      };  // NOLINT
06344     template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<854»; };  // NOLINT
06345     template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; };  // NOLINT
06346     template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; };  // NOLINT
06347     template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, ZPZV<3»; };  // NOLINT
06348     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
06349     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
06350     template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<854»; };  // NOLINT
06351     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
06352     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<308>, ZPZV<719>, ZPZV<854»;
      };  // NOLINT
06353     template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<857»; };  // NOLINT
06354     template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; };  // NOLINT
06355     template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; };  // NOLINT
```

```
06356     template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; };  // NOLINT
06357     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
06358     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
06359     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
06360     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
06361     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
06362     template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<858»; };  // NOLINT
06363     template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; };  // NOLINT
06364     template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; };  // NOLINT
06365     template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<770>, ZPZV<5»; };  // NOLINT
06366     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
06367     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
06368     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
06369     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
06370     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
06371     template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<875»; };  // NOLINT
06372     template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; };  // NOLINT
06373     template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; };  // NOLINT
06374     template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; };  // NOLINT
06375     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
06376     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
06377     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
06378     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
06379     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
06380     template<> struct ConwayPolynomial<881, 1> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<878»; };  // NOLINT
06381     template<> struct ConwayPolynomial<881, 2> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; };  // NOLINT
06382     template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; };  // NOLINT
06383     template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; };  // NOLINT
06384     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
06385     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
06386     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
06387     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
06388     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
06389     template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
      POLYV<ZPZV<1>, ZPZV<881»; };  // NOLINT
06390     template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
      POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; };  // NOLINT
06391     template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; };  // NOLINT
06392     template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; };  // NOLINT
06393     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
06394     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
06395     template<> struct ConwayPolynomial<883, 7> { using ZPZ = aerobus::zpz<883>; using type =
```

```
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<881»; };  // NOLINT
06396     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
06397     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
06398     template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
      POLYV<ZPZV<1>, ZPZV<882»; };  // NOLINT
06399     template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
      POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; };  // NOLINT
06400     template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; };  // NOLINT
06401     template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; };  // NOLINT
06402     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
06403     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
06404     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
06405     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
06406     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
06407     template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<905»; };  // NOLINT
06408     template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»; };  // NOLINT
06409     template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; };  // NOLINT
06410     template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; };  // NOLINT
06411     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
06412     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
06413     template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; };  // NOLINT
06414     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
06415     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
06416     template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
      POLYV<ZPZV<1>, ZPZV<894»; };  // NOLINT
06417     template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
      POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; };  // NOLINT
06418     template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; };  // NOLINT
06419     template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; };  // NOLINT
06420     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
06421     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
06422     template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<894»; };  // NOLINT
06423     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
06424     template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<679>, ZPZV<116>, ZPZV<894»;
      };  // NOLINT
06425     template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<912»; };  // NOLINT
06426     template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; };  // NOLINT
06427     template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; };  // NOLINT
06428     template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; };  // NOLINT
06429     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
06430     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
06431     template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<912»; };  // NOLINT
06432     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
06433     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
```

```
06434     template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<926>>; };  // NOLINT
06435     template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3>>; };  // NOLINT
06436     template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926>>; };  // NOLINT
06437     template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3>>; };  // NOLINT
06438     template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<926>>; };  // NOLINT
06439     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
06440     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
06441     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
06442     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
06443     template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<932>>; };  // NOLINT
06444     template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5>>; };  // NOLINT
06445     template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932>>; };  // NOLINT
06446     template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<23>, ZPZV<585>, ZPZV<5>>; };  // NOLINT
06447     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
06448     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
06449     template<> struct ConwayPolynomial<937, 7> { using ZPZ = aerobus::zpz<937>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<932>>; };  // NOLINT
06450     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
06451     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
06452     template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<939>>; };  // NOLINT
06453     template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2>>; };  // NOLINT
06454     template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939>>; };  // NOLINT
06455     template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2>>; };  // NOLINT
06456     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
06457     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
06458     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
06459     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
06460     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
06461     template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<945>>; };  // NOLINT
06462     template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2>>; };  // NOLINT
06463     template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945>>; };  // NOLINT
06464     template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2>>; };  // NOLINT
06465     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
06466     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
06467     template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<945>>; };  // NOLINT
06468     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
06469     template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<269>, ZPZV<808>, ZPZV<945>>;
      };  // NOLINT
06470     template<> struct ConwayPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<950>>; };  // NOLINT
06471     template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3>>; };  // NOLINT
06472     template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950>>; };  // NOLINT
06473     template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
```

```
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<865>, ZPZV<3»; };  // NOLINT
06474     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
06475     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
06476     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
06477     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
06478     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
06479     template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
    POLYV<ZPZV<1>, ZPZV<962»; };  // NOLINT
06480     template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
    POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; };  // NOLINT
06481     template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; };  // NOLINT
06482     template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; };  // NOLINT
06483     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
06484     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
06485     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
06486     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
06487     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
06488     template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
    POLYV<ZPZV<1>, ZPZV<965»; };  // NOLINT
06489     template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
    POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; };  // NOLINT
06490     template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<965»; };  // NOLINT
06491     template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<527>, ZPZV<6»; };  // NOLINT
06492     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
06493     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
06494     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
06495     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
06496     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
06497     template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
    POLYV<ZPZV<1>, ZPZV<974»; };  // NOLINT
06498     template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
    POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; };  // NOLINT
06499     template<> struct ConwayPolynomial<977, 3> { using ZPZ = aerobus::zpz<977>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; };  // NOLINT
06500     template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<800>, ZPZV<3»; };  // NOLINT
06501     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
06502     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
06503     template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<974»; };  // NOLINT
06504     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
06505     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
06506     template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
    POLYV<ZPZV<1>, ZPZV<978»; };  // NOLINT
06507     template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
    POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; };  // NOLINT
06508     template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; };  // NOLINT
06509     template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; };  // NOLINT
06510     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
06511     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
06512     template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type =
    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<978»; };  // NOLINT
```

```
06513     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
06514     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
06515     template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<985>>; };  // NOLINT
06516     template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6>>; };  // NOLINT
06517     template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985>>; };  // NOLINT
06518     template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6>>; };  // NOLINT
06519     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
06520     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
06521     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
06522     template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<941>, ZPZV<786>, ZPZV<234>, ZPZV<6>>; };  //
      NOLINT
06523     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
06524     template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<990>>; };  // NOLINT
06525     template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7>>; };  // NOLINT
06526     template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990>>; };  // NOLINT
06527     template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7>>; };  // NOLINT
06528     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
06529     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
06530     template<> struct ConwayPolynomial<997, 7> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<990>>; };  // NOLINT
06531     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
06532     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
06533 #endif  // DO_NOT_DOCUMENT
06534 }  // namespace aerobus
06535 #endif  // AEROBUS_CONWAY_IMPORTS
06536
06537 #endif // __INC_AEROBUS__ // NOLINT
```

9.4 src/examples.h File Reference

9.5 examples.h

Go to the documentation of this file.

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

Chapter 10

Examples

10.1 examples/hermite.cpp

How to use aerobus::known_polynomials::hermite_phys polynomials

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

namespace standardlib {
    double H3(double x) {
        return 8 * std::pow(x, 3) - 12 * x;
    }

    double H4(double x) {
        return 16 * std::pow(x, 4) - 48 * x * x + 12;
    }
}

namespace aerobuslib {
    double H3(double x) {
        return 8 * aerobus::pow_scalar<double, 3>(x) - 12 * x;
    }

    double H4(double x) {
        return 16 * aerobus::pow_scalar<double, 4>(x) - 48 * x * x + 12;
    }
}

int main() {
    std::cout << std::hermite(3, 10) << '=' << standardlib::H3(10) << '\n'
              << std::hermite(4, 10) << '=' << standardlib::H4(10) << '\n';
    std::cout << aerobus::known_polynomials::hermite_phys<3>::eval(10) << '=' << aerobuslib::H3(10) << '\n'
              << aerobus::known_polynomials::hermite_phys<4>::eval(10) << '=' << aerobuslib::H4(10) << '\n';
}
```

10.2 examples/custom_taylor.cpp

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

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


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

template<size_t deg>
```

```
using F = aerobus::taylor<aerobus::i64, my_coeff, deg>;

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

10.3 examples/fp16.cu

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

```
// TO compile with nvcc -O3 -std=c++20 -arch=sm_90 fp16.cu
// Beforehand, you need to modify cuda_fp16.h by adding __CUDA_FP16_CONSTEXPR__ to line 5039 (version 12.6)
#include <cstdio>

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

/*
You may want to change int_type to aerobus::i32 (or i64) and float_type to float (resp. double)
*/
using int_type = aerobus::i16;
using float_type = __half2;


constexpr size_t N = 1 « 26;

template<typename T>
struct Expm1Degree;

template<>
struct Expm1Degree<double> {
    static constexpr size_t val = 18;
};

template<>
struct Expm1Degree<float> {
    static constexpr size_t val = 11;
};

template<>
struct Expm1Degree<__half2> {
    static constexpr size_t val = 6;
};

template<>
struct Expm1Degree<__half> {
    static constexpr size_t val = 6;
};

double rand(double min, double max) {
  double range = (max - min);
  double div = RAND_MAX / range;
  return min + (rand() / div);  // NOLINT
}

template<typename T>
struct GetRandT;

template<>
struct GetRandT<double> {
    static double func(double min, double max) {
        return rand(min, max);
    }
};

template<>
struct GetRandT<float> {
    static float func(double min, double max) {
        return (float) rand(min, max);
    }
};

template<>
struct GetRandT<__half2> {
    static __half2 func(double min, double max) {
        return __half2(__float2half((float)rand(min, max)), __float2half((float)rand(min, max)));
    }
};

template<>
```

```
struct GetRandT<__half> {
    static __half func(double min, double max) {
        return __float2half((float)rand(min, max));
    }
};

using EXPM1 = aerobus::expm1<int_type, Expm1Degree<float_type>::val>;


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

__global__ void run(size_t N, float_type* in, float_type* out) {
    for(size_t i = threadIdx.x + blockDim.x * blockIdx.x; i < N; i += blockDim.x * gridDim.x) {
        // fp16 FMA pipeline is quite wide so we need to feed it with a LOT of computations
        out[i] = f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(in[i])))))))))))))))));
    }
}

#define cudaErrorCheck(ans) { gpuAssert((ans), __FILE__, __LINE__); }
inline void gpuAssert(cudaError_t code, const char *file, int line, bool abort=true)
{
   if (code != cudaSuccess)
   {
      fprintf(stderr,"GPUassert: %s %s %d\n", cudaGetErrorString(code), file, line);
      if (abort) exit(code);
   }
}

int main() {
    // configure CUDA devices
    int deviceCount;
    int device = -1;
    int maxProcCount = 0;
    cudaErrorCheck(cudaGetDeviceCount(&deviceCount));
    for(int i = 0; i < deviceCount; ++i) {
        cudaDeviceProp prop;
        cudaErrorCheck(cudaGetDeviceProperties(&prop, i));
        int procCount = prop.multiProcessorCount;
        if(procCount > maxProcCount) {
            maxProcCount = procCount;
            device = i;
        }
    }

    if(device == -1) {
        ::printf("CANNOT FIND CUDA CAPABLE DEVICE -- aborting\n");
        ::abort();
    }

    cudaErrorCheck(cudaSetDevice(device));
    int blockSize;   // The launch configurator returned block size
    int minGridSize; // The minimum grid size needed to achieve the
                     // maximum occupancy for a full device launch

    cudaErrorCheck(cudaOccupancyMaxPotentialBlockSize( &minGridSize, &blockSize, &run, 0, 0));

    ::printf("configure launch bounds to %d-%d\n", minGridSize, blockSize);

    // allocate and populate memory
    float_type *d_in, *d_out;
    cudaErrorCheck(cudaMalloc<float_type>(&d_in, N * sizeof(float_type)));
    cudaErrorCheck(cudaMalloc<float_type>(&d_out, N * sizeof(float_type)));

    float_type *in = reinterpret_cast<float_type*>(malloc(N * sizeof(float_type)));
    float_type *out = reinterpret_cast<float_type*>(malloc(N * sizeof(float_type)));

    for(size_t i = 0; i < N; ++i) {
        in[i] = GetRandT<float_type>::func(-0.01, 0.01);
    }

    cudaErrorCheck(cudaMemcpy(d_in, in, N * sizeof(float_type), cudaMemcpyHostToDevice));

    // execute kernel and get memory back from device
    run<<<minGridSize, blockSize>>>(N, d_in, d_out);
    cudaErrorCheck(cudaPeekAtLastError());
    cudaErrorCheck(cudaMemcpy(out, d_out, N * sizeof(float_type), cudaMemcpyDeviceToHost));

    cudaErrorCheck(cudaFree(d_in));
    cudaErrorCheck(cudaFree(d_out));
}

// Example of generated SASS :

/*
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;
```

```
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R6, R5, 0.5, 0.5 ;
HFMA2 R5, R6, R5, 1, 1 ;
HFMA2.MMA R5, R6, R5, RZ ;
HFMA2 R7, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R7, R5, R7, 0.008331298828125, 0.008331298828125 ;
HFMA2 R7, R5, R7, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R7, R5, R7, 0.1666259765625, 0.1666259765625 ;
HFMA2 R7, R5, R7, 0.5, 0.5 ;
HFMA2.MMA R7, R5, R7, 1, 1 ;
HFMA2 R7, R5, R7, RZ.H0_H0 ;
HFMA2.MMA R5, R7, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R7, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R7, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R7, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R7, R5, 0.5, 0.5 ;
HFMA2 R5, R7, R5, 1, 1 ;
HFMA2.MMA R5, R7, R5, RZ ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125 ;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625 ;
HFMA2 R6, R5, R6, 0.5, 0.5 ;
HFMA2.MMA R6, R5, R6, 1, 1 ;
HFMA2 R6, R5, R6, RZ.H0_H0 ;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R6, R5, 0.5, 0.5 ;
HFMA2 R5, R6, R5, 1, 1 ;
HFMA2.MMA R5, R6, R5, RZ ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125 ;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625 ;
HFMA2 R6, R5, R6, 0.5, 0.5 ;
HFMA2.MMA R6, R5, R6, 1, 1 ;
HFMA2 R6, R5, R6, RZ.H0_H0 ;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R6, R5, 0.5, 0.5 ;
HFMA2 R5, R6, R5, 1, 1 ;
HFMA2.MMA R5, R6, R5, RZ ;
HFMA2 R6, R5, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R6, R5, R6, 0.008331298828125, 0.008331298828125 ;
HFMA2 R6, R5, R6, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R6, R5, R6, 0.1666259765625, 0.1666259765625 ;
HFMA2 R6, R5, R6, 0.5, 0.5 ;
HFMA2.MMA R6, R5, R6, 1, 1 ;
HFMA2 R6, R5, R6, RZ.H0_H0 ;
HFMA2.MMA R5, R6, RZ, 0.0013885498046875, 0.0013885498046875 ;
HFMA2 R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2.MMA R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2 R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2.MMA R5, R6, R5, 0.5, 0.5 ;
HFMA2 R5, R6, R5, 1, 1 ;
HFMA2.MMA R6, R6, R5, RZ ;
HFMA2 R5, R6, RZ.H0_H0, 0.0013885498046875, 0.0013885498046875 ;
HFMA2.MMA R5, R6, R5, 0.008331298828125, 0.008331298828125 ;
HFMA2 R5, R6, R5, 0.041656494140625, 0.041656494140625 ;
HFMA2.MMA R5, R6, R5, 0.1666259765625, 0.1666259765625 ;
HFMA2 R5, R6, R5, 0.5, 0.5 ;
HFMA2.MMA R7, R6, R5, 1, 1 ;
IADD3.X R5, R8, UR11, RZ, P0, !PT ;
IADD3 R3, P0, R2, R3, RZ ;
IADD3.X R0, RZ, R0, RZ, P0, !PT ;
ISETP.GE.U32.AND P0, PT, R3, UR8, PT ;
HFMA2 R7, R6, R7, RZ.H0_H0 ;
```

```
ISETP.GE.U32.AND.EX P0, PT, R0, UR9, PT, P0 ;
STG.E desc[UR6][R4.64], R7 ;
*/
```

10.4 examples/continued_fractions.cpp

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

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

static constexpr double PHI = aerobus::ContinuedFraction<
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1>::val;

static const double phi = (std::sqrt(5.0) + 1.0)/2.0;

int main() {
    std::cout « std::setprecision(15) « "Aerobus  PHI : " « PHI « std::endl;
    std::cout « std::setprecision(15) « "Computed PHI : " « phi « std::endl;
    return 0;
}
```

10.5 examples/modular_arithmetic.cpp

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

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

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

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

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

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

10.6 examples/make_polynomial.cpp

How to build your own sequence of known polynomials, here `Abel polynomials`

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


// let's build Abel polynomials from scratch using Aerobus
// note : it's now integrated in the main library, but still serves as an example

template<typename I = aerobus::i64>
struct AbelHelper {
 private:
    using P = aerobus::polynomial<I>;

 public:
    // to keep recursion working, we need to operate on a*n and not just a
    template<size_t deg, I::inner_type an>
    struct Inner {
```

```
        // abel(n, a) = (x-an) * abel(n-1, a)
        using type = typename aerobus::mul_t<
            typename Inner<deg-1, an>::type,
            typename aerobus::sub_t<typename P::X, typename P::template inject_constant_t<an>>
        >;
    };

    // abel(0, a) = 1
    template<I::inner_type an>
    struct Inner<0, an> {
        using type = P::one;
    };

    // abel(1, a) = X
    template<I::inner_type an>
    struct Inner<1, an> {
        using type = P::X;
    };
};

template<size_t n, auto a, typename I = aerobus::i64>
using AbelPolynomials = typename AbelHelper<I>::template Inner<n, a*n>::type;

using A2_3 = AbelPolynomials<3, 2>;

int main() {
    std::cout << "expected = x^3 - 12 x^2 + 36 x" << std::endl;
    std::cout << "aerobus  = " << A2_3::to_string() << std::endl;
    return 0;
}
```

10.7 examples/polynomials_over_finite_field.cpp

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

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

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

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

10.8 examples/compensated_horner.cpp

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

See also

publication

```
// run with ./generate_comp_horner.sh in this directory
// that will compile and run this sample and plot all the generated data
#include "../src/aerobus.h"

using namespace aerobus;  // NOLINT

constexpr size_t NB_POINTS = 400;

template<typename P, typename T, bool compensated>
DEVICE INLINED T eval(const T& x) {
```

```
    if constexpr (compensated) {
        return P::template compensated_eval<T>(x);
    } else {
        return P::template eval<T>(x);
    }
}

template<typename T>
DEVICE T exact_large(const T& x) {
    return pow_scalar<T, 5>(0.75 - x) * pow_scalar<T, 11>(1 - x);
}

template<typename T>
DEVICE T exact_small(const T& x) {
    return pow_scalar<T, 3>(x - 1);
}

template<typename P, typename T, bool compensated>
void run(T left, T right, const char *file_name, T (*exact)(const T&)) {
    FILE *f = ::fopen(file_name, "w+");
    T step = (right - left) / NB_POINTS;
    T x = left;
    for (size_t i = 0; i <= NB_POINTS; ++i) {
        ::fprintf(f, "%e %e %e\n", x, eval<P, T, compensated>(x), exact(x));
        x += step;
    }
    ::fclose(f);
}

int main() {
    {
        // (0.75 - x)^5 * (1 - x)^11
        using P = mul_t<
            pow_t<pq64, pq64::val<
                typename q64::template inject_constant_t<-1>,
                q64::val<i64::val<3>, i64::val<4>>, 5>,
            pow_t<pq64, pq64::val<typename q64::template inject_constant_t<-1>, typename q64::one>, 11>
        >;
        using FLOAT = double;
        run<P, FLOAT, false>(0.68, 1.15, "plots/large_sample_horner.dat", &exact_large);
        run<P, FLOAT, true>(0.68, 1.15, "plots/large_sample_comp_horner.dat", &exact_large);

        run<P, FLOAT, false>(0.74995, 0.75005, "plots/first_root_horner.dat", &exact_large);
        run<P, FLOAT, true>(0.74995, 0.75005, "plots/first_root_comp_horner.dat", &exact_large);

        run<P, FLOAT, false>(0.9935, 1.0065, "plots/second_root_horner.dat", &exact_large);
        run<P, FLOAT, true>(0.9935, 1.0065, "plots/second_root_comp_horner.dat", &exact_large);
    }
    {
        // (x - 1) ^ 3
        using P = make_int_polynomial_t<i32, 1, -3, 3, -1>;

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

Index