Aerobus

v1.2

Generated by Doxygen 1.9.8

| 1 Introduction | 1 |
|---|----|
| 1.1 HOW TO | 1 |
| 1.1.1 Unit Test | 2 |
| 1.1.2 Benchmarks | 2 |
| 1.2 Structures | 3 |
| 1.2.1 Predefined discrete euclidean domains | 3 |
| 1.2.2 Polynomials | 3 |
| 1.2.3 Known polynomials | 4 |
| 1.2.4 Conway polynomials | 4 |
| 1.2.5 Taylor series | 4 |
| 1.3 Operations | 6 |
| 1.3.1 Field of fractions | 6 |
| 1.3.2 Quotient | 6 |
| 1.4 Misc | 7 |
| 1.4.1 Continued Fractions | 7 |
| 2 Namespace Index | 9 |
| 2.1 Namespace List | 9 |
| 3 Concept Index | 11 |
| 3.1 Concepts | 11 |
| 4 Class Index | 13 |
| 4.1 Class List | 13 |
| 5 File Index | 15 |
| 5.1 File List | 15 |
| 6 Namespace Documentation | 17 |
| 6.1 aerobus Namespace Reference | 17 |
| 6.1.1 Detailed Description | 20 |
| 6.1.2 Typedef Documentation | 21 |
| 6.1.2.1 abs_t | 21 |
| 6.1.2.2 addfractions_t | 21 |
| 6.1.2.3 alternate_t | 21 |
| 6.1.2.4 asin | 21 |
| 6.1.2.5 asinh | 22 |
| 6.1.2.6 atan | 22 |
| 6.1.2.7 atanh | 22 |
| 6.1.2.8 bernoulli_t | 22 |
| 6.1.2.9 combination_t | 23 |
| 6.1.2.10 cos | 23 |
| 6.1.2.11 cosh | 23 |
| 6.1.2.12 E_fraction | 23 |

| | 6.1.2.13 exp | 24 |
|--------|--------------------------------------|----|
| | 6.1.2.14 expm1 | 24 |
| | 6.1.2.15 factorial_t | 24 |
| | 6.1.2.16 fpq32 | 24 |
| | 6.1.2.17 fpq64 | 25 |
| | 6.1.2.18 FractionField | 25 |
| | 6.1.2.19 gcd_t | 25 |
| | 6.1.2.20 geometric_sum | 25 |
| | 6.1.2.21 lnp1 | 25 |
| | 6.1.2.22 make_q32_t | 26 |
| | 6.1.2.23 make_q64_t | 26 |
| | 6.1.2.24 makefraction_t | 26 |
| | 6.1.2.25 mulfractions_t | 26 |
| | 6.1.2.26 pi64 | 27 |
| | 6.1.2.27 PI_fraction | 27 |
| | 6.1.2.28 pow_t | 27 |
| | 6.1.2.29 pq64 | 27 |
| | 6.1.2.30 q32 | 27 |
| | 6.1.2.31 q64 | 28 |
| | 6.1.2.32 sin | 28 |
| | 6.1.2.33 sinh | 28 |
| | 6.1.2.34 SQRT2_fraction | 28 |
| | 6.1.2.35 SQRT3_fraction | 28 |
| | 6.1.2.36 stirling_signed_t | 28 |
| | 6.1.2.37 stirling_unsigned_t | 29 |
| | 6.1.2.38 tan | 29 |
| | 6.1.2.39 tanh | 29 |
| | 6.1.2.40 taylor | 30 |
| | 6.1.2.41 vadd_t | 30 |
| | 6.1.2.42 vmul_t | 30 |
| (| 6.1.3 Function Documentation | 30 |
| | 6.1.3.1 aligned_malloc() | 30 |
| | 6.1.3.2 field() | 31 |
| (| 6.1.4 Variable Documentation | 31 |
| | 6.1.4.1 alternate_v | 31 |
| | 6.1.4.2 bernoulli_v | 31 |
| | 6.1.4.3 combination_v | 32 |
| | 6.1.4.4 factorial_v | 32 |
| 6.2 ae | erobus::internal Namespace Reference | 32 |
| (| 6.2.1 Detailed Description | 35 |
| (| 6.2.2 Typedef Documentation | 35 |
| | 6.2.2.1 make_index_sequence_reverse | 35 |
| | | |

| 6.2.2.2 type_at_t | 35 |
|--|----|
| 6.2.3 Function Documentation | 36 |
| 6.2.3.1 index_sequence_reverse() | 36 |
| 6.2.4 Variable Documentation | 36 |
| 6.2.4.1 is_instantiation_of_v | 36 |
| 6.3 aerobus::known_polynomials Namespace Reference | 36 |
| 6.3.1 Detailed Description | 37 |
| 6.3.2 Typedef Documentation | 37 |
| 6.3.2.1 bernoulli | 37 |
| 6.3.2.2 bernstein | 37 |
| 6.3.2.3 chebyshev_T | 37 |
| 6.3.2.4 chebyshev_U | 38 |
| 6.3.2.5 hermite_phys | 38 |
| 6.3.2.6 hermite_prob | 39 |
| 6.3.2.7 laguerre | 39 |
| 6.3.2.8 legendre | 39 |
| 6.3.3 Enumeration Type Documentation | 40 |
| 6.3.3.1 hermite_kind | 40 |
| 7 Concept Documentation | 41 |
| 7.1 aerobus::IsEuclideanDomain Concept Reference | 41 |
| 7.1.1 Concept definition | 41 |
| 7.1.2 Detailed Description | 41 |
| 7.2 aerobus::IsField Concept Reference | 41 |
| 7.2.1 Concept definition | 41 |
| 7.2.2 Detailed Description | 42 |
| 7.3 aerobus::IsRing Concept Reference | 42 |
| 7.3.1 Concept definition | 42 |
| 7.3.2 Detailed Description | 42 |
| | |
| 8 Class Documentation | 43 |
| 8.1 aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, E > Struct Template Reference | 43 |
| 8.2 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 index > 0)> > Struct Template Reference | 43 |
| 8.2.1 Member Typedef Documentation | 43 |
| 8.2.1.1 type | 43 |
| 8.3 aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference | 44 |
| 8.3.1 Member Typedef Documentation | 44 |
| 8.3.1.1 type | 44 |
| 8.4 aerobus::ContinuedFraction < values > Struct Template Reference | 44 |
| 8.5 aerobus::ContinuedFraction < a0 > Struct Template Reference | 44 |
| 8.5.1 Detailed Description | 44 |

| 8.5.2 Member Typedef Documentation | 5 |
|--|----|
| 8.5.2.1 type | 5 |
| 8.5.3 Member Data Documentation | 5 |
| 8.5.3.1 val | 5 |
| 8.6 aerobus::ContinuedFraction< a0, rest > Struct Template Reference | 5 |
| 8.6.1 Detailed Description | 5 |
| 8.6.2 Member Typedef Documentation | 6 |
| 8.6.2.1 type | 6 |
| 8.6.3 Member Data Documentation | 6 |
| 8.6.3.1 val | 6 |
| 8.7 aerobus::ConwayPolynomial Struct Reference | 6 |
| 8.8 aerobus::i32 Struct Reference | 6 |
| 8.8.1 Detailed Description | .7 |
| 8.8.2 Member Typedef Documentation | 8 |
| 8.8.2.1 add_t | 8 |
| 8.8.2.2 div_t | 8 |
| 8.8.2.3 eq_t | 8 |
| 8.8.2.4 gcd_t | 8 |
| 8.8.2.5 gt_t | 8 |
| 8.8.2.6 inject_constant_t | 8 |
| 8.8.2.7 inject_ring_t | 8 |
| 8.8.2.8 inner_type | 8 |
| 8.8.2.9 lt_t | 8 |
| 8.8.2.10 mod_t | 8 |
| 8.8.2.11 mul_t | 9 |
| 8.8.2.12 one | 9 |
| 8.8.2.13 pos_t | 9 |
| 8.8.2.14 sub_t | 9 |
| 8.8.2.15 zero | 9 |
| 8.8.3 Member Data Documentation | 9 |
| 8.8.3.1 eq_v | 9 |
| 8.8.3.2 is_euclidean_domain | .9 |
| 8.8.3.3 is_field | 0 |
| 8.8.3.4 pos_v | 0 |
| 8.9 aerobus::i64 Struct Reference | 0 |
| 8.9.1 Detailed Description | 1 |
| 8.9.2 Member Typedef Documentation | 1 |
| 8.9.2.1 add_t | 1 |
| 8.9.2.2 div_t | 1 |
| 8.9.2.3 eq_t | 1 |
| 8.9.2.4 gcd_t | 2 |
| 8.9.2.5 gt_t | 2 |

| 8.9.2.6 inject_constant_t | 52 |
|---|----|
| 8.9.2.7 inject_ring_t | 52 |
| 8.9.2.8 inner_type | 52 |
| 8.9.2.9 lt_t | 52 |
| 8.9.2.10 mod_t | 52 |
| 8.9.2.11 mul_t | 53 |
| 8.9.2.12 one | 53 |
| 8.9.2.13 pos_t | 53 |
| 8.9.2.14 sub_t | 53 |
| 8.9.2.15 zero | 53 |
| 8.9.3 Member Data Documentation | 53 |
| 8.9.3.1 eq_v | 53 |
| 8.9.3.2 gt_v | 53 |
| 8.9.3.3 is_euclidean_domain | 54 |
| 8.9.3.4 is_field | 54 |
| 8.9.3.5 lt_v | 54 |
| 8.9.3.6 pos_v | 54 |
| 8.10 aerobus::is_prime $<$ n $>$ Struct Template Reference | 54 |
| 8.10.1 Detailed Description | 54 |
| 8.10.2 Member Data Documentation | 55 |
| 8.10.2.1 value | 55 |
| 8.11 aerobus::polynomial $<$ Ring $>$ Struct Template Reference | 55 |
| 8.11.1 Detailed Description | 56 |
| 8.11.2 Member Typedef Documentation | 57 |
| 8.11.2.1 add_t | 57 |
| 8.11.2.2 derive_t | 57 |
| 8.11.2.3 div_t | 57 |
| 8.11.2.4 eq_t | 57 |
| 8.11.2.5 gcd_t | 58 |
| 8.11.2.6 gt_t | 58 |
| 8.11.2.7 inject_constant_t | 58 |
| 8.11.2.8 inject_ring_t | 58 |
| 8.11.2.9 lt_t | 59 |
| 8.11.2.10 mod_t | 59 |
| 8.11.2.11 monomial_t | 59 |
| 8.11.2.12 mul_t | 59 |
| 8.11.2.13 one | 60 |
| 8.11.2.14 pos_t | 60 |
| 8.11.2.15 simplify_t | 60 |
| 8.11.2.16 sub_t | 60 |
| 8.11.2.17 X | 61 |
| 8.11.2.18 zero | 61 |

| 8.11.3 Member Data Documentation | 61 |
|---|----|
| 8.11.3.1 is_euclidean_domain | 61 |
| 8.11.3.2 is_field | 61 |
| 8.11.3.3 pos_v | 61 |
| 8.12 aerobus::type_list< Ts >::pop_front Struct Reference | 61 |
| 8.12.1 Detailed Description | 62 |
| 8.12.2 Member Typedef Documentation | 62 |
| 8.12.2.1 tail | 62 |
| 8.12.2.2 type | 62 |
| 8.13 aerobus::Quotient $<$ Ring, X $>$ Struct Template Reference | 62 |
| 8.13.1 Detailed Description | 63 |
| 8.13.2 Member Typedef Documentation | 64 |
| 8.13.2.1 add_t | 64 |
| 8.13.2.2 div_t | 64 |
| 8.13.2.3 eq_t | 64 |
| 8.13.2.4 inject_constant_t | 64 |
| 8.13.2.5 inject_ring_t | 65 |
| 8.13.2.6 mod_t | 65 |
| 8.13.2.7 mul_t | 65 |
| 8.13.2.8 one | 65 |
| 8.13.2.9 pos_t | 66 |
| 8.13.2.10 zero | 66 |
| 8.13.3 Member Data Documentation | 66 |
| 8.13.3.1 eq_v | 66 |
| 8.13.3.2 is_euclidean_domain | 66 |
| 8.13.3.3 pos_v | 66 |
| 8.14 aerobus::type_list< Ts >::split< index > Struct Template Reference | 67 |
| 8.14.1 Detailed Description | 67 |
| 8.14.2 Member Typedef Documentation | 67 |
| 8.14.2.1 head | 67 |
| 8.14.2.2 tail | 67 |
| 8.15 aerobus::type_list< Ts > Struct Template Reference | 68 |
| 8.15.1 Detailed Description | 68 |
| 8.15.2 Member Typedef Documentation | 69 |
| 8.15.2.1 at | 69 |
| 8.15.2.2 concat | 69 |
| 8.15.2.3 insert | 69 |
| 8.15.2.4 push_back | 69 |
| 8.15.2.5 push_front | 70 |
| 8.15.2.6 remove | 70 |
| 8.15.3 Member Data Documentation | 70 |
| 8.15.3.1 length | 70 |

| 8.16 aerobus::type_list<> Struct Reference | 70 |
|---|----|
| 8.16.1 Detailed Description | 71 |
| 8.16.2 Member Typedef Documentation | 71 |
| 8.16.2.1 concat | 71 |
| 8.16.2.2 insert | 71 |
| 8.16.2.3 push_back | 71 |
| 8.16.2.4 push_front | 71 |
| 8.16.3 Member Data Documentation | 72 |
| 8.16.3.1 length | 72 |
| 8.17 aerobus::i32::val $<$ x $>$ Struct Template Reference | 72 |
| 8.17.1 Detailed Description | 72 |
| 8.17.2 Member Typedef Documentation | 73 |
| 8.17.2.1 enclosing_type | 73 |
| 8.17.2.2 is_zero_t | 73 |
| 8.17.3 Member Function Documentation | 73 |
| 8.17.3.1 eval() | 73 |
| 8.17.3.2 get() | 73 |
| 8.17.3.3 to_string() | 74 |
| 8.17.4 Member Data Documentation | 74 |
| 8.17.4.1 v | 74 |
| 8.18 aerobus::i64::val < x > Struct Template Reference | 74 |
| 8.18.1 Detailed Description | 74 |
| 8.18.2 Member Typedef Documentation | 75 |
| 8.18.2.1 enclosing_type | 75 |
| 8.18.2.2 is_zero_t | 75 |
| 8.18.3 Member Function Documentation | 75 |
| 8.18.3.1 eval() | 75 |
| 8.18.3.2 get() | 75 |
| 8.18.3.3 to_string() | 76 |
| 8.18.4 Member Data Documentation | 76 |
| 8.18.4.1 v | 76 |
| 8.19 aerobus::polynomial < Ring >::val < coeffN, coeffs > Struct Template Reference | 76 |
| 8.19.1 Detailed Description | 77 |
| 8.19.2 Member Typedef Documentation | 77 |
| 8.19.2.1 aN | 77 |
| 8.19.2.2 coeff_at_t | 77 |
| 8.19.2.3 enclosing_type | 78 |
| 8.19.2.4 is_zero_t | 78 |
| 8.19.2.5 strip | 78 |
| 8.19.3 Member Function Documentation | 78 |
| 8.19.3.1 eval() | 78 |
| 8.19.3.2 to_string() | 79 |

| 8.19.4 Member Data Documentation | . 79 |
|---|------|
| 8.19.4.1 degree | . 79 |
| 8.19.4.2 is_zero_v | . 79 |
| 8.20 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference | . 79 |
| 8.20.1 Detailed Description | . 79 |
| 8.20.2 Member Typedef Documentation | . 80 |
| 8.20.2.1 type | . 80 |
| 8.21 aerobus::zpz::val< x > Struct Template Reference | . 80 |
| 8.21.1 Member Typedef Documentation | . 80 |
| 8.21.1.1 enclosing_type | . 80 |
| 8.21.1.2 is_zero_t | . 81 |
| 8.21.2 Member Function Documentation | . 81 |
| 8.21.2.1 eval() | . 81 |
| 8.21.2.2 get() | . 81 |
| 8.21.2.3 to_string() | . 81 |
| 8.21.3 Member Data Documentation | . 81 |
| 8.21.3.1 v | . 81 |
| 8.22 aerobus::polynomial < Ring >::val < coeffN > Struct Template Reference | . 81 |
| 8.22.1 Detailed Description | . 82 |
| 8.22.2 Member Typedef Documentation | . 82 |
| 8.22.2.1 aN | . 82 |
| 8.22.2.2 coeff_at_t | . 83 |
| 8.22.2.3 enclosing_type | . 83 |
| 8.22.2.4 is_zero_t | . 83 |
| 8.22.2.5 strip | . 83 |
| 8.22.3 Member Function Documentation | . 83 |
| 8.22.3.1 eval() | . 83 |
| 8.22.3.2 to_string() | . 83 |
| 8.22.4 Member Data Documentation | . 84 |
| 8.22.4.1 degree | . 84 |
| 8.22.4.2 is_zero_v | . 84 |
| 8.23 aerobus::zpz $<$ p $>$ Struct Template Reference | . 84 |
| 8.23.1 Detailed Description | . 85 |
| 8.23.2 Member Typedef Documentation | . 85 |
| 8.23.2.1 add_t | . 85 |
| 8.23.2.2 div_t | . 86 |
| 8.23.2.3 eq_t | . 86 |
| 8.23.2.4 gcd_t | . 86 |
| 8.23.2.5 gt_t | . 87 |
| 8.23.2.6 inject_constant_t | . 87 |
| 8.23.2.7 inner_type | . 87 |
| 8.23.2.8 lt_t | . 87 |

| | 8.23.2.9 mod_t | 87 |
|----|----------------------------------|-----|
| | 8.23.2.10 mul_t | 88 |
| | 8.23.2.11 one | 88 |
| | 8.23.2.12 pos_t | 88 |
| | 8.23.2.13 sub_t | 88 |
| | 8.23.2.14 zero | 89 |
| | 8.23.3 Member Data Documentation | 89 |
| | 8.23.3.1 eq_v | 89 |
| | 8.23.3.2 gt_v | 89 |
| | 8.23.3.3 is_euclidean_domain | 89 |
| | 8.23.3.4 is_field | 89 |
| | 8.23.3.5 lt_v | 90 |
| | 8.23.3.6 pos_v | 90 |
| 0 | File Documentation | 91 |
| 9 | | 91 |
| | 9.2 src/aerobus.h File Reference | |
| | 9.3 aerobus.h | |
| | 9.5 delobus.ii | 91 |
| 10 |) Examples | 175 |
| | 10.1 QuotientRing | 175 |
| | 10.2 type_list | 175 |
| | 10.3 i32::template | 175 |
| | 10.4 i32::add_t | 176 |
| | 10.5 i32::sub_t | 176 |
| | 10.6 i32::mul_t | 176 |
| | 10.7 i32::div_t | 176 |
| | 10.8 i32::gt_t | 177 |
| | 10.9 i32::eq_t | 177 |
| | 10.10 i32::eq_v | 177 |
| | 10.11 i32::gcd_t | 177 |
| | 10.12 i32::pos_t | 178 |
| | 10.13 i32::pos_v | 178 |
| | 10.14 i64::template | 178 |
| | 10.15 i64::add_t | |
| | 10.16 i64::sub_t | 179 |
| | 10.17 i64::mul_t | 179 |
| | 10.18 i64::div_t | 179 |
| | 10.19 i64::mod_t | 179 |
| | 10.20 i64::gt_t | 180 |
| | 10.21 i64::lt_t | 180 |
| | 10.22 i64::lt_v | 180 |
| | 10.23 i64::eq_t | 180 |

| Inc | dex | 185 |
|-----|----------------------------------|-----|
| | 10.33 E_fraction::val | 183 |
| | 10.32 PI_fraction::val | |
| | 10.31 aerobus::ContinuedFraction | 182 |
| | 10.30 FractionField | 182 |
| | 10.29 q32::add_t | 182 |
| | 10.28 polynomial | 182 |
| | 10.27 i64::pos_v | 181 |
| | 10.26 i64::pos_t | 181 |
| | 10.25 i64::gcd_t | 181 |
| | 10.24 i64::eq_v | 181 |

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

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

2 Introduction

1.1.1 Unit Test

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

Move to the top directory then:

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

Terminal should write:

100% tests passed, 0 tests failed out of 48

Alternate way:

make tests

From top directory.

1.1.2 Benchmarks

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

In addition of Cmake and compiler, install OpenMP. Then move to top directory:

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

results on my laptop:

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

1.2 Structures 3

1.2 Structures

1.2.1 Predefined discrete euclidean domains

Aerobus predefines several simple euclidean domains, such as :

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

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

```
• add_t : a type (specialization of val), representing addition between two values
```

- 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) \{\ldots\}
```

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

4 Introduction

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 (\neg 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 $Fraction \leftarrow Field < Integers > .$

They can be used and evaluated:

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

Exposed functions are:

- exp
- $\bullet \ \mathrm{expm1} \ e^x 1$
- lnp1 ln(x+1)
- geom $\frac{1}{1-x}$
- sin

1.2 Structures 5

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

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

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

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

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

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

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

```
compute_expml(unsigned long, double const*, double*):
          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]
                  ymm11, QWORD PTR .LC7[rip]
 vbroadcastsd
          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]
                   ymm6, QWORD PTR .LC17[rip]
  vbroadcastsd
                   ymm5, QWORD PTR .LC19[rip]
 vbroadcastsd
  vbroadcastsd
                   ymm4, QWORD PTR .LC21[rip]
```

6 Introduction

```
ymm3, QWORD PTR .LC23[rip]
 vbroadcastsd
                 ymm2, QWORD PTR .LC25[rip]
 vbroadcastsd
.L3:
 vmovupd ymm15, YMMWORD PTR [rsi+rax]
 vmovapd ymm0, ymm15
                 ymm0, ymm14, ymm1
 vfmadd132pd
 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
         rax, 32
 cmp
         rcx, rax
         .L3
 ine
 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 $\ \, \text{quotient ring } R/X \ \, \text{where X is a principal}$ ideal generated by some element, as we know this kind of ideal is two-sided as long as R is commutative (and we assume it is).

```
For example, if we want R to be \mathbb{Z} represented as aerobus::i64, we can express arithmetic modulo 17 using: using namespace aerobus; using \text{ZpZ} = \text{Quotient} < \text{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 7

1.4 Misc

1.4.1 Continued Fractions

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

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

8 Introduction

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

| aerobus | |
|--|----|
| Main namespace for all publicly exposed types or functions | 17 |
| aerobus::internal | |
| Internal implementations, subject to breaking changes without notice | 32 |
| aerobus::known_polynomials | |
| Families of well known polynomials such as Hermite or Bernstein | 36 |

10 Namespace Index

Concept Index

3.1 Concepts

Here is a list of all concepts with brief descriptions:

| aerobus::IsEuclideanDomain | |
|---|----|
| Concept to express R is an euclidean domain | 41 |
| aerobus::IsField | |
| Concept to express R is a field | 41 |
| aerobus::IsRing | |
| Concept to express R is a Ring | 42 |

12 Concept Index

Class Index

4.1 Class List

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

| aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, E > | 43 |
|---|----|
| $aerobus::polynomial < Ring >::val < coeffN >::coeff_at < index, std::enable_if_t < (index < 0 index > 0) > > 43$ | |
| aerobus::polynomial< Ring >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)>> | 44 |
| aerobus::ContinuedFraction < values > | 44 |
| aerobus::ContinuedFraction < a0 > | |
| Specialization for only one coefficient, technically just 'a0' | 44 |
| aerobus::ContinuedFraction< a0, rest > | |
| Specialization for multiple coefficients (strictly more than one) | 45 |
| aerobus::ConwayPolynomial | 46 |
| aerobus::i32 | |
| 32 bits signed integers, seen as a algebraic ring with related operations | 46 |
| aerobus::i64 | |
| 64 bits signed integers, seen as a algebraic ring with related operations | 50 |
| aerobus::is_prime< n > | |
| Checks if n is prime | 54 |
| aerobus::polynomial < Ring > | 55 |
| aerobus::type_list< Ts >::pop_front | |
| Removes types from head of the list | 61 |
| 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 | 62 |
| aerobus::type_list< Ts >::split< index > | |
| Splits list at index | 67 |
| aerobus::type_list< Ts > | |
| Empty pure template struct to handle type list | 68 |
| aerobus::type_list<> | |
| Specialization for empty type list | 70 |
| aerobus::i32::val < x > | |
| Values in i32, again represented as types | 72 |
| aerobus::i64::val< x > | |
| Values in i64 | 74 |
| aerobus::polynomial< Ring >::val< coeffN, coeffs > | |
| Values (seen as types) in polynomial ring | 76 |
| aerobus::Quotient< Ring, X >::val< V > | |
| Projection values in the quotient ring | 79 |

| 4 | Class Index |
|---|-------------|
| | |

| aerobus::zpz::val< x > | 80 |
|--|----|
| aerobus::polynomial < Ring >::val < coeffN > | |
| Specialization for constants | 81 |
| aerobus::zpz | 84 |

File Index

| 5.1 | File | List |
|-----|------|------|
|-----|------|------|

| Here is a list of all files | with | brie | f de | scrip | otio | ns: | | | | | | | | | | | | | |
|-----------------------------|------|------|------|-------|------|-----|--|--|--|--|--|--|------|--|--|--|------|--|----|
| src/aerobus.h | | | | | | | | | | | | | | | | | | | 91 |

16 File Index

Namespace Documentation

6.1 aerobus Namespace Reference

main namespace for all publicly exposed types or functions

Namespaces

· namespace internal

internal implementations, subject to breaking changes without notice

namespace known_polynomials

families of well known polynomials such as Hermite or Bernstein

Classes

- struct ContinuedFraction
- struct ContinuedFraction < a0 >

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

struct ContinuedFraction < a0, rest... >

specialization for multiple coefficients (strictly more than one)

- · struct ConwayPolynomial
- struct i32

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

• struct i64

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

• struct is_prime

checks if n is prime

- · struct polynomial
- struct Quotient

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

struct type_list

Empty pure template struct to handle type list.

struct type list<>

specialization for empty type list

struct zpz

Concepts

· concept IsRing

Concept to express R is a Ring.

• concept IsEuclideanDomain

Concept to express R is an euclidean domain.

· concept IsField

Concept to express R is a field.

Typedefs

```
    template<typename T, typename A, typename B>

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

    template<typename... vals>

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

    template<typename val >

  using abs t = std::conditional t < val::enclosing type::template pos v < val >, val, typename val::enclosing ←
  _type::template sub_t< typename val::enclosing_type::zero, val > >
      computes absolute value of 'val' val must be a 'value' in a Ring satisfying 'IsEuclideanDomain' concept
• template<typename Ring >
  using FractionField = typename internal::FractionFieldImpl< Ring >::type
using q32 = FractionField < i32 >
      32 bits rationals rationals with 32 bits numerator and denominator

    using fpq32 = FractionField< polynomial< q32 >>

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

 using q64 = FractionField < i64 >

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

    using fpq64 = FractionField< polynomial< q64 > >

      polynomial with 64 bits rational coefficients

    template<typename Ring , typename v1 , typename v2 >

  using makefraction_t = typename FractionField< Ring >::template val< v1, v2 >
      helper type: the rational V1/V2 in the field of fractions of Ring
• template<int64_t p, int64_t q>
  using make_q64_t = typename q64::template simplify_t< typename q64::val< i64::inject_constant_t< p>,
  i64::inject_constant_t< q >>>
```

using make_q32_t = typename q32::template simplify_t< typename q32::val< i32::inject_constant_t< p>,

helper type: make a fraction from numerator and denominator

helper type: make a fraction from numerator and denominator

• template<int32_t p, int32_t q>

 $i32::inject_constant_t < q >> >$

```
• template<typename Ring , typename v1 , typename v2 >
  using addfractions_t = typename FractionField< Ring >::template add t< v1, v2 >
     helper type : adds two fractions

    template<typename Ring , typename v1 , typename v2 >

  using mulfractions t = typename FractionField < Ring >::template mul t < v1, v2 >
     helper type: multiplies two fractions
• template<typename T , size_t i>
  using factorial_t = typename internal::factorial < T, i >::type
     computes factorial(i), as type
• template<typename T , size_t k, size_t n>
  using combination_t = typename internal::combination < T, k, n >::type
     computes binomial coefficient (k among n) as type
template<typename T, size_t n>
  using bernoulli_t = typename internal::bernoulli < T, n >::type
      nth bernoulli number as type in T
• template<typename T , int k>
  using alternate_t = typename internal::alternate < T, k >::type
     (-1)^{\wedge}k as type in T

    template<typename T , int n, int k>

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

    template<typename T, auto p, auto n>

  using pow t = typename internal::pow < T, p, n >::type
     p^{\wedge}n (as 'val' type in T)
• template<typename T, template< typename, size_t index > typename coeff_at, size_t deg>
  using taylor = typename internal::make taylor impl< T, coeff at, internal::make index sequence reverse<
  deg+1 > > :: type

    template<typename Integers, size t deg>

  using exp = taylor< Integers, internal::exp_coeff, deg >
• 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 >

    template<typename Integers , size_t deg>

  using sinh = taylor < Integers, internal::sh_coeff, deg >
• template<typename Integers , size_t deg>
  using cosh = taylor< Integers, internal::cosh_coeff, deg >
     \cosh(x) hyperbolic cosine

    template<typename Integers , size_t deg>

  using cos = taylor < Integers, internal::cos_coeff, deg >
```

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

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

• using E_fraction = ContinuedFraction < 2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1 >
approximation of \sqrt{2}
• using SQRT3_fraction = ContinuedFraction < 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 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 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></ring> |
| v2 | belongs to FranctionField <ring></ring> |

6.1.2.3 alternate_t

```
\label{template} $$ template < typename T , int k > $$ using aerobus::alternate_t = typedef typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: type $$ typename internal::alternate < T, k > :: typename internal::alternate < T, typename internal::alternate <
```

(-1)[^]k as type in T

Template Parameters

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

6.1.2.4 asin

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

Template Parameters

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

6.1.2.5 asinh

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

Template Parameters

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

6.1.2.6 atan

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

Template Parameters

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

6.1.2.7 atanh

```
template<typename Integers , size_t deg> using aerobus::atanh = typedef taylor<Integers, internal::atanh_coeff, deg> \operatorname{arctanh}(x) arc hyperbolic tangent
```

Template Parameters

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

6.1.2.8 bernoulli_t

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

nth bernoulli number as type in T

Template Parameters

| T | Ring type (i64) |
|---|-----------------|
| n | |

6.1.2.9 combination_t

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

computes binomial coefficient (k among n) as type

Template Parameters

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

6.1.2.10 cos

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

Template Parameters

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

6.1.2.11 cosh

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

 $\cosh(x)$ hyperbolic cosine

Template Parameters

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

6.1.2.12 E_fraction

```
using aerobus::E_fraction = typedef ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1,
```

```
1, 10, 1, 1, 12, 1, 1, 14, 1, 1>
```

6.1.2.13 exp

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

Template Parameters

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

6.1.2.14 expm1

```
template<typename Integers , size_t deg> using aerobus::expm1 = typedef typename polynomial<FractionField<Integers>>::template sub_ \leftarrow t< exp<Integers, deg>, typename polynomial<FractionField<Integers>>::one> e^x-1
```

Template Parameters

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

6.1.2.15 factorial_t

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

computes factorial(i), as type

Template Parameters

| T | Ring type (e.g. i32) |
|---|----------------------|
| i | |

6.1.2.16 fpq32

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

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

6.1.2.17 fpq64

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

polynomial with 64 bits rational coefficients

6.1.2.18 FractionField

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

6.1.2.19 gcd t

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

computes the greatest common divisor or A and B

Template Parameters

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

6.1.2.20 geometric_sum

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

Template Parameters

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

6.1.2.21 Inp1

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

Template Parameters

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

6.1.2.22 make_q32_t

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

helper type: make a fraction from numerator and denominator

Template Parameters

| р | numerator |
|---|-------------|
| q | denominator |

6.1.2.23 make_q64_t

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

helper type: make a fraction from numerator and denominator

Template Parameters

| р | numerator |
|---|-------------|
| q | denominator |

6.1.2.24 makefraction t

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

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

Template Parameters

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

6.1.2.25 mulfractions_t

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

helper type: multiplies two fractions

Template Parameters

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

6.1.2.26 pi64

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

polynomial with 64 bits integers coefficients

6.1.2.27 Pl_fraction

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

6.1.2.28 pow_t

```
template<typename T , auto p, auto n> using aerobus::pow_t = typedef typename internal::pow<T, p, n>::type p^{\Lambda}n \; (as\; 'val'\; type\; in\; T)
```

Template Parameters

| T | (some ring type, such as aerobus::i64) |
|---|--|
| р | (from T::inner_type, such as int64_t) |
| n | (from T::inner_type, such as int64_t) |

6.1.2.29 pq64

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

polynomial with 64 bits rationals coefficients

6.1.2.30 q32

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

32 bits rationals rationals with 32 bits numerator and denominator

6.1.2.31 q64

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

64 bits rationals rationals with 64 bits numerator and denominator

6.1.2.32 sin

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

Template Parameters

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

6.1.2.33 sinh

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

Template Parameters

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

6.1.2.34 SQRT2_fraction

6.1.2.35 SQRT3_fraction

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

6.1.2.36 stirling_signed_t

```
template<typename T , int n, int k>
using aerobus::stirling_signed_t = typedef typename internal::stirling_helper<T, n, k>::type
Stirling number of first king (signed) - as types.
```

Template Parameters

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

6.1.2.37 stirling_unsigned_t

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

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

Template Parameters

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

6.1.2.38 tan

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

tan(x) tangent

Template Parameters

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

6.1.2.39 tanh

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

tanh(x) hyperbolic tangent

Template Parameters

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

6.1.2.40 taylor

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

Template Parameters

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

6.1.2.41 vadd_t

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

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

Template Parameters

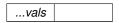


6.1.2.42 vmul_t

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

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

Template Parameters



6.1.3 Function Documentation

6.1.3.1 aligned_malloc()

'portable' aligned allocation of count elements of type T

Template Parameters

| T the type of elements to store | , |
|---------------------------------|---|
|---------------------------------|---|

Parameters

| count | the number of elements |
|-----------|------------------------|
| alignment | boundary |

6.1.3.2 field()

```
brief Conway polynomials tparam p characteristic of the aerobus::field ( $\operatorname{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]
```

$(-1)^{\wedge}$ k as value from T

Template Parameters

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

6.1.4.2 bernoulli_v

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

nth bernoulli number as value in FloatType

Template Parameters

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

6.1.4.3 combination_v

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

computes binomial coefficients (k among n) as value

Template Parameters

| Т | (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

| Т | (aerobus::i64 for example) |
|---|----------------------------|
| i | |

6.2 aerobus::internal Namespace Reference

internal implementations, subject to breaking changes without notice

Classes

- $\bullet \; \mathsf{struct} \, \underline{\mathsf{FractionField}}$
- struct FractionField< Ring, std::enable if t< Ring::is euclidean domain > >
- · struct _is_prime
- struct $_{\bf is_prime}$ < 0, i >
- struct _is_prime< 1, i >
- struct $_{is}$ _prime< 2, i >
- struct $_{\mbox{is_prime}}<$ 3, i >
- struct $_{\mathbf{is_prime}}$ < 5, \mathbf{i} >
- struct $_{\bf is_prime}$ < 7, i >
- struct _is_prime< n, i, std::enable_if_t<(n !=2 &&n !=3 &&n % 2 !=0 &&n % 3==0)>>
- struct _is_prime< n, i, std::enable_if_t<(n !=2 &&n % 2==0)> >
- struct _is_prime< n, i, std::enable_if_t<(n % i==0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&i *i > n)>> >
- struct _is_prime< n, i, std::enable_if_t<(n %(i+2) !=0 &&n % i !=0 &&n >=9 &&n % 3 !=0 &&n % 2 !=0 &&n % 2 !=0 &&n % 2 !=0

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

• struct _is_prime< n, i, std::enable_if_t<(n >=9 &&i *i > n)> >
· struct alternate

 struct alternate < T, k, std::enable if t < k % 2 !=0 > >

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

    struct asin coeff

    struct asin_coeff_helper

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

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

· struct asinh coeff

    struct asinh coeff helper

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

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

    struct atan coeff helper

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

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

    struct atanh_coeff

    struct atanh_coeff_helper

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

    struct bernoulli

    struct bernoulli < T, 0 >

    struct bernoulli coeff

    struct bernoulli_helper

    struct bernoulli_helper< T, accum, m, m >

    struct bernstein helper

• struct bernstein helper < 0, 0 >
• struct bernstein_helper< i, m, std::enable_if_t<(m > 0) &&(i > 0) &&(i < m)> >
struct bernstein_helper< i, m, std::enable_if_t<(m > 0) &&(i==0)> >
struct bernstein_helper< i, m, std::enable_if_t<(m > 0) &&(i==m)>>
• struct chebyshev_helper

    struct chebyshev helper< 1, 0 >

    struct chebyshev_helper< 1, 1 >

    struct chebyshev helper< 2, 0 >

    struct chebyshev_helper< 2, 1 >

    struct combination

    struct combination helper

    struct combination_helper< T, 0, n >

• struct combination helper < T, k, n, std::enable if t<(n >=0 &&k >(n/2) &&k > 0)> >
struct combination_helper< T, k, n, std::enable_if_t<(n >=0 &&k<=(n/2) &&k > 0)> >

    struct cos coeff

    struct cos_coeff_helper

struct cos_coeff_helper< T, i, std::enable_if_t<(i &1)==0 >>
struct cos_coeff_helper< T, i, std::enable_if_t<(i &1)==1 >>
· struct cosh coeff

    struct cosh_coeff_helper

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

    struct exp_coeff

    struct factorial

    struct factorial < T, 0 >

struct factorial< T, x, std::enable_if_t<(x > 0)>>

    struct FractionFieldImpl
```

```
    struct FractionFieldImpl< Field, std::enable_if_t< Field::is_field >>

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

    struct gcd

     greatest common divisor computes the greatest common divisor exposes it in gcd<A, B>::type as long as Ring type
     is an integral domain

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

    struct geom_coeff

· struct hermite helper

    struct hermite helper< 0, known polynomials::hermite kind::physicist >

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

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

    struct hermite_helper< 1, known_polynomials::hermite_kind::probabilist >

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

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

· struct insert h
· struct is instantiation of

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

    struct laguerre helper

struct laguerre_helper< 0 >

    struct laguerre helper< 1 >

    struct legendre helper

    struct legendre_helper< 0 >

struct legendre_helper< 1 >

    struct Inp1_coeff

• struct Inp1_coeff< T, 0 >

    struct make taylor impl

    struct make_taylor_impl< T, coeff_at, std::integer_sequence< size_t, ls... >>

    struct pop front h

· struct pow

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

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

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

    struct remove h

· struct sh coeff
• struct sh_coeff_helper

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

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

    struct sin_coeff

    struct sin coeff helper

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

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

· struct split h

    struct split_h< 0, L1, L2 >

    struct stirling helper

    struct stirling_helper< T, 0, 0 >

• struct stirling_helper< T, 0, n, std::enable_if_t<(n > 0)> >

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

    struct stirling_helper< T, n, k, std::enable_if_t<(k > 0) &&(n > 0)> >

· struct tan coeff

    struct tan coeff helper

struct tan_coeff_helper< T, i, std::enable_if_t<(i % 2) !=0 >>

    struct tan_coeff_helper< T, i, std::enable_if_t<(i % 2)==0 >>

· struct tanh coeff
```

· struct tanh_coeff_helper

- struct tanh_coeff_helper< T, i, std::enable_if_t<(i % 2) !=0 >>
- struct tanh_coeff_helper< T, i, std::enable_if_t<(i % 2)==0 >>
- · struct type at
- struct type_at< 0, T, Ts... >
- · struct vadd
- struct vadd< v1 >
- struct vadd< v1, vals... >
- struct vmul
- struct vmul< v1 >
- struct vmul< v1, vals... >

Typedefs

```
    template<size_t i, typename... Ts>
    using type_at_t = typename type_at< i, Ts... >::type
```

```
    template < std::size_t N >
        using make_index_sequence_reverse = decltype(index_sequence_reverse(std::make_index_sequence < N >{}))
```

Functions

template<std::size_t... ls>
 constexpr auto index_sequence_reverse (std::index_sequence< ls... > const &) -> decltype(std::index_
 sequence< sizeof...(ls) - 1U - ls... >{})

Variables

template<template< typename... > typename TT, typename T >
 constexpr bool is instantiation_of_v = is_instantiation_of<TT, T>::value

6.2.1 Detailed Description

internal implementations, subject to breaking changes without notice

6.2.2 Typedef Documentation

6.2.2.1 make_index_sequence_reverse

```
template<std::size_t N>
using aerobus::internal::make_index_sequence_reverse = typedef decltype(index_sequence_reverse(std
::make_index_sequence<N>{}))
```

6.2.2.2 type_at_t

```
template<size_t i, typename... Ts>
using aerobus::internal::type_at_t = typedef typename type_at<i, Ts...>::type
```

6.2.3 Function Documentation

6.2.3.1 index_sequence_reverse()

6.2.4 Variable Documentation

6.2.4.1 is_instantiation_of_v

```
template<template< typename... > typename TT, typename T >
constexpr bool aerobus::internal::is_instantiation_of_v = is_instantiation_of<TT, T>::value
[inline], [constexpr]
```

6.3 aerobus::known_polynomials Namespace Reference

families of well known polynomials such as Hermite or Bernstein

Typedefs

```
template<size_t deg>
  using chebyshev T = typename internal::chebyshev helper< 1, deg >::type
     Chebyshev polynomials of first kind.
template<size t deg>
  using chebyshev_U = typename internal::chebyshev_helper< 2, deg >::type
     Chebyshev polynomials of second kind.
template<size_t deg>
  using laguerre = typename internal::laguerre_helper< deg >::type
     Laguerre polynomials.
• template<size_t deg>
  using hermite prob = typename internal::hermite helper< deg, hermite kind::probabilist >::type
     Hermite polynomials - probabilist form.
template<size_t deg>
  using hermite_phys = typename internal::hermite_helper< deg, hermite_kind::physicist >::type
     Hermite polynomials - physicist form.
• template<size ti, size tm>
  using bernstein = typename internal::bernstein_helper< i, m >::type
     Bernstein polynomials.

    template<size_t deg>

  using legendre = typename internal::legendre helper< deg >::type
     Legendre polynomials.
template<size_t deg>
  using bernoulli = taylor< i64, internal::bernoulli_coeff< deg >::template inner, deg >
     Bernoulli polynomials.
```

Enumerations

• enum hermite_kind { probabilist , physicist }

6.3.1 Detailed Description

families of well known polynomials such as Hermite or Bernstein

6.3.2 Typedef Documentation

6.3.2.1 bernoulli

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

Bernoulli polynomials.

See also

```
See in Wikipedia
```

Template Parameters

```
deg degree of polynomial
```

6.3.2.2 bernstein

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

Bernstein polynomials.

See also

```
See in Wikipedia
```

Template Parameters

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

6.3.2.3 chebyshev_T

template<size_t deg>

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

Chebyshev polynomials of first kind.

See also

See in Wikipedia

Template Parameters

deg degree of polynomial

6.3.2.4 chebyshev_U

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

Chebyshev polynomials of second kind.

See also

See in Wikipedia

Template Parameters

deg degree of polynomial

6.3.2.5 hermite_phys

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

Hermite polynomials - physicist form.

See also

See in Wikipedia

Template Parameters

deg degree of polynomial

6.3.2.6 hermite_prob

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

Hermite polynomials - probabilist form.

See also

```
See in Wikipedia
```

Template Parameters

```
deg degree of polynomial
```

6.3.2.7 laguerre

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

Laguerre polynomials.

See also

```
See in Wikipedia
```

Template Parameters

```
deg degree of polynomial
```

6.3.2.8 legendre

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

Legendre polynomials.

See also

```
See in Wikipedia
```

Template Parameters

```
deg degree of polynomial
```

6.3.3 Enumeration Type Documentation

6.3.3.1 hermite_kind

enum aerobus::known_polynomials::hermite_kind

Enumerator

| probabilist | |
|-------------|--|
| physicist | |

Chapter 7

Concept Documentation

7.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

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

7.1.1 Concept definition

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

7.1.2 Detailed Description

Concept to express R is an euclidean domain.

7.2 aerobus::IsField Concept Reference

Concept to express R is a field.

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

7.2.1 Concept definition

7.2.2 Detailed Description

Concept to express R is a field.

7.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring.

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

7.3.1 Concept definition

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

7.3.2 Detailed Description

Concept to express R is a Ring.

Chapter 8

Class Documentation

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

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

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

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

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

Public Types

• using type = typename Ring::zero

8.2.1 Member Typedef Documentation

8.2.1.1 type

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

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

• src/aerobus.h

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

#include <aerobus.h>

Public Types

using type = aN

8.3.1 Member Typedef Documentation

8.3.1.1 type

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

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

• src/aerobus.h

8.4 aerobus::ContinuedFraction< values > Struct Template Reference

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

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

• src/aerobus.h

8.5 aerobus::ContinuedFraction < a0 > Struct Template Reference

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

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

Public Types

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

Static Public Attributes

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

8.5.1 Detailed Description

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

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

Template Parameters

```
a0 an integer int64_t
```

8.5.2 Member Typedef Documentation

8.5.2.1 type

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

8.5.3 Member Data Documentation

8.5.3.1 val

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

represented value as double

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

· src/aerobus.h

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

specialization for multiple coefficients (strictly more than one)

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

Public Types

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

Static Public Attributes

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

8.6.1 Detailed Description

```
\label{lem:continued} \begin{split} & template {<} int64\_t \ a0, \ int64\_t... \ rest{>} \\ & struct \ aerobus::ContinuedFraction {<} \ a0, \ rest... \ > \end{split}
```

specialization for multiple coefficients (strictly more than one)

Template Parameters

| a0 | integer (int64_t) |
|------|-------------------|
| rest | integers |
| | (int64_t) |

8.6.2 Member Typedef Documentation

8.6.2.1 type

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

represented value as aerobus::q64

8.6.3 Member Data Documentation

8.6.3.1 val

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

represented value as double

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

• src/aerobus.h

8.7 aerobus::ConwayPolynomial Struct Reference

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

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

• src/aerobus.h

8.8 aerobus::i32 Struct Reference

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

#include <aerobus.h>

Classes

• struct val values in i32, again represented as types

Public Types

```
• using inner_type = int32_t
using zero = val< 0 >
     constant zero
using one = val< 1 >
     constant one

    template<auto x>

  using inject_constant_t = val< static_cast< int32_t >(x)>

    template<typename v >

 using inject_ring_t = v

    template<typename v1 , typename v2 >

  using add_t = typename add< v1, v2 >::type
• template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type
• template<typename v1 , typename v2 >
  using mul_t = typename mul < v1, v2 >::type
• template<typename v1 , typename v2 >
  using div_t = typename div < v1, v2 >::type

    template<typename v1 , typename v2 >

  using mod_t = typename remainder < v1, v2 >::type
     modulus operator yields v1 % v2 for example : i32::mod_t<i32::val<7>, i32::val<2>>
• template<typename v1 , typename v2 >
  using gt_t = typename gt < v1, v2 >::type
• template<typename v1 , typename v2 >
  using It_t = typename It< v1, v2 >::type
• template<typename v1 , typename v2 >
  using eq_t = typename eq< v1, v2 >::type
• template<typename v1 , typename v2 >
  using gcd_t = gcd_t < i32, v1, v2 >

    template<typename v >

  using pos t = typename pos< v >::type
```

Static Public Attributes

```
    static constexpr bool is_field = false
        integers are not a field
    static constexpr bool is_euclidean_domain = true
        integers are an euclidean domain
    template<typename v1 , typename v2 >
        static constexpr bool eq_v = eq_t<v1, v2>::value
    template<typename v >
        static constexpr bool pos_v = pos_t<v>::value
```

8.8.1 Detailed Description

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

8.8.2 Member Typedef Documentation

```
8.8.2.1 add_t
template<typename v1 , typename v2 >
using aerobus::i32::add_t = typename add<v1, v2>::type
8.8.2.2 div t
template<typename v1 , typename v2 >
using aerobus::i32::div_t = typename div<v1, v2>::type
8.8.2.3 eq t
template<typename v1 , typename v2 >
using aerobus::i32::eq_t = typename eq<v1, v2>::type
8.8.2.4 gcd_t
template<typename v1 , typename v2 >
using aerobus::i32::gcd_t = gcd_t<i32, v1, v2>
8.8.2.5 gt_t
template<typename v1 , typename v2 >
using aerobus::i32::gt_t = typename gt<v1, v2>::type
8.8.2.6 inject_constant_t
template < auto x >
using aerobus::i32::inject_constant_t = val<static_cast<int32_t>(x)>
8.8.2.7 inject_ring_t
template < typename v >
using aerobus::i32::inject_ring_t = v
8.8.2.8 inner_type
using aerobus::i32::inner_type = int32_t
8.8.2.9 lt_t
template<typename v1 , typename v2 >
using aerobus::i32::lt_t = typename lt<v1, v2>::type
8.8.2.10 mod_t
template<typename v1 , typename v2 >
using aerobus::i32::mod_t = typename remainder<v1, v2>::type
```

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

Template Parameters

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

8.8.2.11 mul_t

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

8.8.2.12 one

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

constant one

8.8.2.13 pos_t

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

8.8.2.14 sub_t

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

8.8.2.15 zero

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

constant zero

8.8.3 Member Data Documentation

8.8.3.1 eq_v

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

8.8.3.2 is_euclidean_domain

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

integers are an euclidean domain

8.8.3.3 is_field

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

integers are not a field

8.8.3.4 pos_v

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

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

• src/aerobus.h

8.9 aerobus::i64 Struct Reference

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

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

Classes

struct val

values in i64

Public Types

```
using inner_type = int64_t
```

type for actual values

template<auto x>

```
using inject_constant_t = val< static_cast< int64_t >(x)>
```

template<typename v >

using inject_ring_t = v

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

using zero = val< 0 >

constant zero

• using one = val< 1 >

constant one

• template<typename v1 , typename v2 >

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

template<typename v1 , typename v2 >

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

• template<typename v1 , typename v2 >

using mul t = typename mul < v1, v2 >::type

template<typename v1 , typename v2 >

using div_t = typename div < v1, v2 >::type

```
template<typename v1, typename v2 > using mod_t = typename remainder< v1, v2 >::type
template<typename v1, typename v2 > using gt_t = typename gt< v1, v2 >::type
template<typename v1, typename v2 > using lt_t = typename lt< v1, v2 >::type
template<typename v1, typename v2 > using eq_t = typename eq< v1, v2 >::type
template<typename v1, typename v2 > using eq_t = typename eq< v1, v2 >::type
template<typename v1, typename v2 > using gcd_t = gcd_t < i64, v1, v2 >
template<typename v > using pos_t = typename pos< v >::type
```

Static Public Attributes

```
    static constexpr bool is_field = false
        integers are not a field
    static constexpr bool is_euclidean_domain = true
        integers are an euclidean domain
```

```
    template < typename v1, typename v2 >
    static constexpr bool gt_v = gt_t < v1, v2 > ::value
    strictly greater operator yields v1 > v2 as boolean value
```

```
    template<typename v1, typename v2 > static constexpr bool lt_v = lt_t<v1, v2>::value
    template<typename v1, typename v2 > static constexpr bool eq_v = eq_t<v1, v2>::value
    template<typename v > static constexpr bool pos_v = pos_t<v>::value
```

8.9.1 Detailed Description

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

8.9.2 Member Typedef Documentation

8.9.2.1 add_t

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

8.9.2.2 div_t

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

8.9.2.3 eq_t

template<typename v1 , typename v2 >
```

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

8.9.2.4 gcd_t

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

8.9.2.5 gt t

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

8.9.2.6 inject_constant_t

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

8.9.2.7 inject_ring_t

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

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

Template Parameters

```
v a value in i64
```

8.9.2.8 inner_type

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

type for actual values

8.9.2.9 It t

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

8.9.2.10 mod_t

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

8.9.2.11 mul_t

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

8.9.2.12 one

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

constant one

8.9.2.13 pos t

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

8.9.2.14 sub_t

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

8.9.2.15 zero

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

constant zero

8.9.3 Member Data Documentation

8.9.3.1 eq_v

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

8.9.3.2 gt_v

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

strictly greater operator yields v1 > v2 as boolean value

Template Parameters

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

8.9.3.3 is_euclidean_domain

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

integers are an euclidean domain

8.9.3.4 is field

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

integers are not a field

8.9.3.5 It v

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

8.9.3.6 pos_v

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

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

• src/aerobus.h

8.10 aerobus::is_prime< n > Struct Template Reference

checks if n is prime

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

Static Public Attributes

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

8.10.1 Detailed Description

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

checks if n is prime

Template Parameters

```
n
```

8.10.2 Member Data Documentation

8.10.2.1 value

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

true iff n is prime

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

· src/aerobus.h

8.11 aerobus::polynomial < Ring > Struct Template Reference

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

Classes

struct val

values (seen as types) in polynomial ring

struct val< coeffN >

specialization for constants

Public Types

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

constant zero

• using one = val< typename Ring::one >

constant one

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

generator

template<typename P >

using simplify_t = typename simplify< P >::type

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

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

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

adds two polynomials

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

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

substraction of two polynomials

```
• template<typename v1 , typename v2 >
  using mul_t = typename mul < v1, v2 >::type
     multiplication of two polynomials

    template<typename v1 , typename v2 >

  using eq_t = typename eq_helper< v1, v2 >::type
     equality operator
• template<typename v1 , typename v2 >
  using lt_t = typename lt_helper< v1, v2 >::type
     strict less operator
• template<typename v1 , typename v2 >
  using gt_t = typename gt_helper< v1, v2 >::type
     strict greater operator

    template<typename v1 , typename v2 >

  using div t = typename div < v1, v2 >::q type
     division operator

    template<typename v1 , typename v2 >

  using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type
     modulo operator
• template<typename coeff , size_t deg>
  using monomial t = typename monomial < coeff, deg >::type
     monomial : coeff X^{\wedge} deg
• template<typename v >
  using derive t = typename derive helper< v >::type
     derivation operator
• template<typename v >
  using pos_t = typename Ring::template pos_t < typename v::aN >
     checks for positivity (an > 0)

    template<typename v1 , typename v2 >

  using gcd_t = std::conditional_t < Ring::is_euclidean_domain, typename make_unit < gcd_t < polynomial <
  Ring >, v1, v2 > ::type, void >
     greatest common divisor of two polynomials

    template<auto x>

  using inject constant t = val< typename Ring::template inject constant t < x > >

    template<typename v >

  using inject_ring_t = val< v >
```

Static Public Attributes

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

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

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

8.11.1 Detailed Description

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

polynomial with coefficients in Ring Ring must be an integral domain

8.11.2 Member Typedef Documentation

8.11.2.1 add t

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

adds two polynomials

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.2 derive_t

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

derivation operator

Template Parameters



8.11.2.3 div_t

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

division operator

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.4 eq_t

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

equality operator

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.5 gcd t

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

greatest common divisor of two polynomials

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.6 gt_t

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

strict greater operator

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.7 inject_constant_t

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

8.11.2.8 inject_ring_t

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

8.11.2.9 lt_t

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

strict less operator

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.10 mod_t

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

modulo operator

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.11 monomial_t

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

$monomial: coeff \ X^{\wedge} deg$

Template Parameters

| coeff | |
|-------|--|
| deg | |

8.11.2.12 mul_t

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

multiplication of two polynomials

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.13 one

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

constant one

8.11.2.14 pos t

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

checks for positivity (an > 0)

Template Parameters



8.11.2.15 simplify_t

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

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

Template Parameters



8.11.2.16 sub_t

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

substraction of two polynomials

Template Parameters

| v1 | |
|----|--|
| v2 | |

8.11.2.17 X

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

8.11.2.18 zero

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

constant zero

8.11.3 Member Data Documentation

8.11.3.1 is_euclidean_domain

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

8.11.3.2 is_field

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

8.11.3.3 pos_v

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

positivity operator

Template Parameters

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

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

• src/aerobus.h

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

removes types from head of the list

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

Public Types

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

8.12.1 Detailed Description

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

removes types from head of the list

8.12.2 Member Typedef Documentation

8.12.2.1 tail

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

remaining types in parent list when front is removed

8.12.2.2 type

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

type that was previously head of the list

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

src/aerobus.h

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

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

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

Classes

struct val

projection values in the quotient ring

Public Types

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

     zero value
using one = val< typename Ring::one >
• 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
  using inject_constant_t = val< typename Ring::template inject_constant_t < x > >

    template<typename v >

  using inject ring t = val< v >
```

Static Public Attributes

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

8.13.1 Detailed Description

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

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

Template Parameters

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

8.13.2 Member Typedef Documentation

8.13.2.1 add t

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

addition operator

Template Parameters

| v1 | a value in quotient ring |
|----|--------------------------|
| v2 | a value in quotient ring |

8.13.2.2 div_t

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

division operator

Template Parameters

| v1 | a value in quotient ring |
|----|--------------------------|
| v2 | a value in quotient ring |

8.13.2.3 eq_t

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

equality operator (as type)

Template Parameters

| V | 1 | a value in quotient ring |
|----|---|--------------------------|
| V2 | 2 | a value in quotient ring |

8.13.2.4 inject constant t

```
template<typename Ring , typename X >
```

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

8.13.2.5 inject_ring_t

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

8.13.2.6 mod_t

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

modulus operator

Template Parameters

| v1 | a value in quotient ring |
|----|--------------------------|
| v2 | a value in quotient ring |

8.13.2.7 mul_t

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

substraction operator

Template Parameters

| v1 | a value in quotient ring |
|----|--------------------------|
| v2 | a value in quotient ring |

8.13.2.8 one

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

one

8.13.2.9 pos_t

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

positivity operator always true

Template Parameters

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

8.13.2.10 zero

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

zero value

8.13.3 Member Data Documentation

8.13.3.1 eq_v

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

addition operator (as boolean value)

Template Parameters

| v1 | a value in quotient ring |
|----|--------------------------|
| v2 | a value in quotient ring |

8.13.3.2 is_euclidean_domain

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

8.13.3.3 pos_v

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

Template Parameters

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

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

· src/aerobus.h

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

```
splits list at index
```

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

Public Types

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

8.14.1 Detailed Description

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

8.14.2 Member Typedef Documentation

8.14.2.1 head

index

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

8.14.2.2 tail

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

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

src/aerobus.h

8.15 aerobus::type_list< Ts > Struct Template Reference

Empty pure template struct to handle type list.

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

Classes

struct pop_front
 removes types from head of the list
 struct split

splits list at index

Public Types

```
template<typename T >
  using push_front = type_list< T, Ts... >
     Adds T to front of the list.
template<size_t index>
  using at = internal::type_at_t< index, Ts... >
     returns type at index
• template<typename T >
  using push_back = type_list< Ts..., T >
     pushes T at the tail of the list
• template<typename U >
  using concat = typename concat_h< U >::type
     concatenates two list into one
• template<typename T , size_t index>
  using insert = typename internal::insert h < index, type list < Ts... >, T >::type
     inserts type at index
template<size_t index>
  using remove = typename internal::remove_h< index, type_list< Ts... >>::type
     removes type at index
```

Static Public Attributes

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

8.15.1 Detailed Description

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

Empty pure template struct to handle type list.

8.15.2 Member Typedef Documentation

8.15.2.1 at

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

returns type at index

Template Parameters

8.15.2.2 concat

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

concatenates two list into one

Template Parameters



8.15.2.3 insert

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

inserts type at index

Template Parameters

| index | |
|-------|--|
| T | |

8.15.2.4 push_back

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

pushes T at the tail of the list

Template Parameters

| T |
|---|
|---|

8.15.2.5 push_front

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

Adds T to front of the list.

Template Parameters



8.15.2.6 remove

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

removes type at index

Template Parameters

```
index
```

8.15.3 Member Data Documentation

8.15.3.1 length

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

length of list

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

• src/aerobus.h

8.16 aerobus::type_list<> Struct Reference

specialization for empty type list

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

Public Types

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

Static Public Attributes

• static constexpr size_t length = 0

8.16.1 Detailed Description

specialization for empty type list

8.16.2 Member Typedef Documentation

8.16.2.1 concat

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

8.16.2.2 insert

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

8.16.2.3 push_back

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

8.16.2.4 push_front

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

8.16.3 Member Data Documentation

8.16.3.1 length

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

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

• src/aerobus.h

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

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

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

Public Types

```
• using enclosing_type = i32
```

Enclosing ring type.

using is_zero_t = std::bool_constant< x==0 >

is value zero

Static Public Member Functions

```
    template<typename valueType >
    static constexpr valueType get ()
```

cast x into valueType

• static std::string to_string ()

string representation of value

 template < typename valueRing > static constexpr valueRing eval (const valueRing &v)

cast x into valueRing

Static Public Attributes

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

8.17.1 Detailed Description

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

values in i32, again represented as types

Template Parameters

```
x an actual integer
```

8.17.2 Member Typedef Documentation

8.17.2.1 enclosing_type

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

Enclosing ring type.

8.17.2.2 is_zero_t

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

is value zero

8.17.3 Member Function Documentation

8.17.3.1 eval()

cast x into valueRing

Template Parameters

```
valueRing double for example
```

8.17.3.2 get()

```
template<iint32_t x>
template<typename valueType >
static constexpr valueType aerobus::i32::val< x >::get ( ) [inline], [static], [constexpr]
```

cast x into valueType

Template Parameters

```
valueType | double for example
```

8.17.3.3 to_string()

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

8.17.4 Member Data Documentation

8.17.4.1 v

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

actual value stored in val type

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

· src/aerobus.h

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

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

Public Types

```
    using enclosing_type = i64

            enclosing ring type

    using is_zero_t = std::bool_constant< x==0 >

            is value zero
```

Static Public Member Functions

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

Static Public Attributes

static constexpr int64_t v = x
 actual value

8.18.1 Detailed Description

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

Template Parameters

```
x an actual integer
```

8.18.2 Member Typedef Documentation

8.18.2.1 enclosing_type

```
template<iint64_t x>
using aerobus::i64::val< x >::enclosing_type = i64
enclosing ring type
```

8.18.2.2 is_zero_t

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

is value zero

8.18.3 Member Function Documentation

8.18.3.1 eval()

cast value in valueRing

Template Parameters

```
valueRing (double for example)
```

8.18.3.2 get()

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

cast value in valueType

Template Parameters

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

8.18.3.3 to_string()

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

8.18.4 Member Data Documentation

8.18.4.1 v

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

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

• src/aerobus.h

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

```
values (seen as types) in polynomial ring
#include <aerobus.h>
```

Public Types

```
    using enclosing_type = polynomial < Ring >
        enclosing ring type
    using aN = coeffN
        heavy weight coefficient (non zero)
```

using strip = val< coeffs... >

remove largest coefficient

using is_zero_t = std::bool_constant<(degree==0) &&(aN::is_zero_t::value)>

true_type if polynomial is constant zero

template<size_t index>

```
using \ coeff\_at\_t = typename \ coeff\_at < index > ::type
```

type of coefficient at index

Static Public Member Functions

```
    static std::string to_string ()
    get a string representation of polynomial
```

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

evaluates polynomial seen as a function operating on ValueRing

Static Public Attributes

```
• static constexpr size_t degree = sizeof...(coeffs)
```

degree of the polynomial

• static constexpr bool is_zero_v = is_zero_t::value

true if polynomial is constant zero

8.19.1 Detailed Description

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

values (seen as types) in polynomial ring

Template Parameters

| coeffN | high degree coefficient |
|--------|---------------------------|
| coeffs | lower degree coefficients |

8.19.2 Member Typedef Documentation

8.19.2.1 aN

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

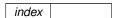
heavy weight coefficient (non zero)

8.19.2.2 coeff_at_t

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

type of coefficient at index

Template Parameters



8.19.2.3 enclosing_type

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

8.19.2.4 is_zero_t

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

true_type if polynomial is constant zero

8.19.2.5 strip

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

remove largest coefficient

8.19.3 Member Function Documentation

8.19.3.1 eval()

evaluates polynomial seen as a function operating on ValueRing

Template Parameters

```
valueRing usually float or double
```

Parameters

```
x value
```

Returns

P(x)

8.19.3.2 to_string()

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

get a string representation of polynomial

Returns

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

8.19.4 Member Data Documentation

8.19.4.1 degree

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

degree of the polynomial

8.19.4.2 is_zero_v

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

true if polynomial is constant zero

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

• src/aerobus.h

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

projection values in the quotient ring

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

Public Types

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

8.20.1 Detailed Description

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

projection values in the quotient ring

Template Parameters

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

8.20.2 Member Typedef Documentation

8.20.2.1 type

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

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

· src/aerobus.h

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

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

Public Types

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

Static Public Member Functions

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

Static Public Attributes

static constexpr int32_t v = x % p
 actual value

8.21.1 Member Typedef Documentation

8.21.1.1 enclosing type

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

enclosing ring type

8.21.1.2 is_zero_t

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

8.21.2 Member Function Documentation

8.21.2.1 eval()

8.21.2.2 get()

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

8.21.2.3 to_string()

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

8.21.3 Member Data Documentation

8.21.3.1 v

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

actual value

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

src/aerobus.h

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

specialization for constants

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

Classes

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

Public Types

```
    using enclosing_type = polynomial < Ring >
        enclosing ring type
    using aN = coeffN
    using strip = val < coeffN >
    using is_zero_t = std::bool_constant < aN::is_zero_t::value >
```

template<size_t index>
 using coeff_at_t = typename coeff_at< index >::type

Static Public Member Functions

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

Static Public Attributes

```
    static constexpr size_t degree = 0
        degree
    static constexpr bool is zero v = is zero t::value
```

8.22.1 Detailed Description

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

specialization for constants

Template Parameters
```

coeffN

8.22.2 Member Typedef Documentation

8.22.2.1 aN

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

8.22.2.2 coeff_at_t

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

8.22.2.3 enclosing_type

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

8.22.2.4 is_zero_t

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

8.22.2.5 strip

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

8.22.3 Member Function Documentation

8.22.3.1 eval()

8.22.3.2 to_string()

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

8.22.4 Member Data Documentation

8.22.4.1 degree

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

degree
```

8.22.4.2 is_zero_v

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

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

• src/aerobus.h

8.23 aerobus::zpz Struct Template Reference

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

Classes

struct val

Public Types

```
• using inner_type = int32_t
template<auto x>
 using inject_constant_t = val< static_cast< int32_t >(x)>
using zero = val< 0 >
• using one = val< 1 >

    template<typename v1 , typename v2 >

 using add_t = typename add< v1, v2 >::type
     addition operator
• template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type
     substraction operator
• template<typename v1 , typename v2 >
  using mul_t = typename mul < v1, v2 >::type
     multiplication operator
• template<typename v1 , typename v2 >
  using div_t = typename div < v1, v2 >::type
     division operator
```

```
• template<typename v1 , typename v2 >
  using mod_t = typename remainder < v1, v2 >::type
     modulo operator

    template<typename v1 , typename v2 >

  using gt_t = typename gt < v1, v2 >::type
     strictly greater operator (type)

    template<typename v1 , typename v2 >

  using It_t = typename It < v1, v2 >::type
     strictly smaller operator (type)

    template<typename v1 , typename v2 >

  using eq_t = typename eq< v1, v2 >::type
      equality operator (type)
• template<typename v1 , typename v2 >
  using gcd_t = gcd_t < i32, v1, v2 >
     greatest common divisor
template<typename v1 >
  using pos_t = typename pos< v1 >::type
     positivity operator (type)
```

Static Public Attributes

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

8.23.1 Detailed Description

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

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

8.23.2 Member Typedef Documentation

8.23.2.1 add t

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

Template Parameters

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

8.23.2.2 div_t

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

division operator

Template Parameters

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

8.23.2.3 eq_t

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

equality operator (type)

Template Parameters

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

8.23.2.4 gcd_t

```
template<iint32_t p>
template<typename v1 , typename v2 >
using aerobus::zpz::gcd_t = gcd_t<i32, v1, v2>
```

greatest common divisor

Template Parameters

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

8.23.2.5 gt_t

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

strictly greater operator (type)

Template Parameters

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

8.23.2.6 inject_constant_t

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

8.23.2.7 inner_type

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

8.23.2.8 It t

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

strictly smaller operator (type)

Template Parameters

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

8.23.2.9 mod_t

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

modulo operator

Template Parameters

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

8.23.2.10 mul_t

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

multiplication operator

Template Parameters

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

8.23.2.11 one

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

8.23.2.12 pos_t

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

positivity operator (type)

Template Parameters

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

8.23.2.13 sub_t

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

substraction operator

Template Parameters

| | v1 | a value in zpz::val |
|---|----|---------------------|
| ſ | v2 | a value in zpz::val |

8.23.2.14 zero

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

8.23.3 Member Data Documentation

8.23.3.1 eq_v

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

equality operator (booleanvalue)

Template Parameters

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

8.23.3.2 gt v

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

strictly greater operator (booleanvalue)

Template Parameters

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

8.23.3.3 is_euclidean_domain

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

8.23.3.4 is_field

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

8.23.3.5 lt_v

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

strictly smaller operator (booleanvalue)

Template Parameters

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

8.23.3.6 pos_v

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

positivity operator (boolean value)

Template Parameters

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

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

• src/aerobus.h

Chapter 9

File Documentation

9.1 README.md File Reference

9.2 src/aerobus.h File Reference

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

9.3 aerobus.h

Go to the documentation of this file.

```
00001 // -*- lsst-c++ -*-
00002 #ifndef __INC_AEROBUS__ // NOLINT
00003 #define __INC_AEROBUS__
00004
00005 #include <cstdint>
00006 #include <cstddef>
00007 #include <cstring>
00008 #include <type_traits>
00009 #include <utility>
00010 #include <algorithm>
00011 #include <functional>
00012 #include <string>
00013 #include <concepts> // NOLINT
00014 #include <array>
00015
00018 #ifdef _MSC_VER
00019 #define ALIGNED(x) __declspec(align(x))
00020 #define INLINED __forceinline
00021 #else
00022 #define ALIGNED(x) __attribute__((aligned(x)))
00023 #define INLINED __attribute__((always_inline)) inline
00024 #endif
00025
00027
00029
```

92 File Documentation

```
00031
00032 // aligned allocation
00033 namespace aerobus {
00040
          template<typename T>
00041
           T* aligned_malloc(size_t count, size_t alignment) {
               #ifdef _MSC_VER
return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
00042
00043
00044
00045
               return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
00046
               #endif
00047
00048 } // namespace aerobus
00049
00050 // concepts
00051 namespace aerobus {
00053
          template <typename R>
           concept IsRing = requires {
00054
00055
               typename R::one;
               typename R::zero;
00057
               typename R::template add_t<typename R::one, typename R::one>;
00058
               typename R::template sub_t<typename R::one, typename R::one>;
00059
               typename R::template mul_t<typename R::one, typename R::one>;
00060
          };
00061
00063
           template <typename R>
           concept IsEuclideanDomain = IsRing<R> && requires {
00064
00065
               typename R::template div_t<typename R::one, typename R::one>;
00066
               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>;
00067
00068
00069
               typename R::template pos t<typename R::one>;
00070
00071
               R::template pos_v<typename R::one> == true;
00072
               // typename R::template gt_t<typename R::one, typename R::zero>;
00073
               R::is_euclidean_domain == true;
00074
00075
00077
           template<typename R>
00078
           concept IsField = IsEuclideanDomain<R> && requires {
00079
             R::is_field == true;
08000
00081 } // namespace aerobus
00082
00083 // utilities
00084 namespace aerobus {
00085
          namespace internal {
00086
               template<template<typename...> typename TT, typename T>
00087
               struct is_instantiation_of : std::false_type { };
00088
               template<template<ttypename...> typename TT, typename... Ts>
struct is_instantiation_of<TT, TT<Ts...» : std::true_type { };</pre>
00089
00090
00091
00092
               template<template<typename...> typename TT, typename T>
00093
               inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
00094
00095
               template <int64_t i, typename T, typename... Ts>
00096
               struct type_at {
00097
                   static_assert(i < sizeof...(Ts) + 1, "index out of range");</pre>
00098
                   using type = typename type_at<i - 1, Ts...>::type;
00099
               };
00100
00101
               template <typename T, typename... Ts> struct type_at<0, T, Ts...> {
00102
                   using type = T;
00103
00104
               template <size_t i, typename... Ts>
using type_at_t = typename type_at<i, Ts...>::type;
00105
00106
00107
00108
00109
               template<size_t n, size_t i, typename E = void>
00110
               struct _is_prime {};
00111
00112
               template<size t i>
               struct _{is\_prime<0, i> \{}
00113
                   static constexpr bool value = false;
00114
00115
00116
00117
               template<size_t i>
00118
               struct _is_prime<1, i> {
00119
                  static constexpr bool value = false;
00120
00121
00122
               template<size_t i>
00123
               struct _is_prime<2, i> {
00124
                   static constexpr bool value = true;
00125
00126
```

9.3 aerobus.h

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

94 File Documentation

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

9.3 aerobus.h

```
template<typename Ring, typename X>
          requires IsRing<Ring>
00325
00326
          struct Quotient {
00329
              template <typename V>
00330
              struct val {
              public:
00331
00332
                  using type = abs_t<typename Ring::template mod_t<V, X>>;
00333
00334
00336
              using zero = val<typename Ring::zero>;
00337
00339
              using one = val<tvpename Ring::one>;
00340
00344
              template<typename v1, typename v2>
00345
              using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
00346
00350
              template<typename v1, typename v2>
00351
              using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
00352
00356
              template<typename v1, typename v2>
00357
              using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
00358
00362
              template<typename v1, typename v2> \,
00363
              using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
00364
00368
              template<typename v1, typename v2>
              using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
00369
00370
00374
              template<typename v1, typename v2>
              static constexpr bool eq_v = Ring::template eq_t<typename v1::type, typename v2::type>::value;
00375
00376
00380
              template<typename v1>
00381
              using pos_t = std::true_type;
00382
00386
              template<typename v>
00387
              static constexpr bool pos_v = pos_t < v > :: value;
00388
              static constexpr bool is_euclidean_domain = true;
00391
00397
              template<auto x>
00398
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
00399
00405
              template<typename v>
00406
              using inject_ring_t = val<v>;
00407
00408 }
        // namespace aerobus
00409
00410 // type_list
00411 namespace aerobus {
         template <typename... Ts>
00413
00414
          struct type_list;
00415
00416
          namespace internal {
00417
              template <typename T, typename... Us>
00418
              struct pop_front_h {
                  using tail = type_list<Us...>;
using head = T;
00419
00420
00421
              };
00422
00423
              template <size_t index, typename L1, typename L2>
00424
              struct split_h {
00425
               private:
00426
                  static_assert(index <= L2::length, "index ouf of bounds");</pre>
                  using a = typename L2::pop_front::type;
00427
00428
                  using b = typename L2::pop_front::tail;
00429
                  using c = typename L1::template push_back<a>;
00430
00431
               public:
00432
                  using head = typename split_h<index - 1, c, b>::head;
                  using tail = typename split_h<index - 1, c, b>::tail;
00433
00434
00435
              template <typename L1, typename L2>
struct split_h<0, L1, L2> {
    using head = L1;
00436
00437
00438
                  using tail = L2;
00439
00440
00441
00442
              template <size_t index, typename L, typename T>
              struct insert_h {
00443
                  static_assert(index <= L::length, "index ouf of bounds");
00444
00445
                  using s = typename L::template split<index>;
00446
                  using left = typename s::head;
00447
                  using right = typename s::tail;
00448
                  using 11 = typename left::template push_back<T>;
00449
                  using type = typename ll::template concat<right>;
00450
              };
```

96 File Documentation

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

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

```
template<typename v1, typename v2>
00697
              using gt_t = typename gt<v1, v2>::type;
00698
00704
              template<typename v1, typename v2>
00705
              using lt_t = typename lt<v1, v2>::type;
00706
00712
              template<typename v1, typename v2>
00713
              using eq_t = typename eq<v1, v2>::type;
00714
00719
              template<typename v1, typename v2>
00720
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00721
              template<typename v1, typename v2>
using gcd_t = gcd_t<i32, v1, v2>;
00727
00728
00729
00734
              template<typename v>
00735
              using pos_t = typename pos<v>::type;
00736
              template<typename v>
00742
              static constexpr bool pos_v = pos_t<v>::value;
00743
00744 } // namespace aerobus
00745
00746 // i64
00747 namespace aerobus {
00749
        struct i64 {
00751
              using inner_type = int64_t;
00754
              template<int64_t x>
00755
              struct val {
                  using enclosing_type = i64;
00757
                  static constexpr int64_t v = x;
00759
00760
00763
                  template<typename valueType>
00764
                  static constexpr valueType get() {
00765
                       return static_cast<valueType>(x);
00766
00767
00769
                  using is_zero_t = std::bool_constant<x == 0>;
00770
00772
                  static std::string to_string() {
00773
                       return std::to_string(x);
00774
                  }
00775
00778
                  template<typename valueRing>
00779
                  static constexpr valueRing eval(const valueRing& v) {
00780
                       return static_cast<valueRing>(x);
00781
00782
              };
00783
00787
              template<auto x>
00788
              using inject_constant_t = val<static_cast<int64_t>(x)>;
00789
00794
              template<typename v>
00795
              using inject_ring_t = v;
00796
00798
              using zero = val<0>;
00800
              using one = val<1>;
00802
              static constexpr bool is_field = false;
00804
              static constexpr bool is_euclidean_domain = true;
00805
00806
           private:
              template<typename v1, typename v2>
00807
80800
              struct add {
00809
                  using type = val<v1::v + v2::v>;
00810
              } ;
00811
00812
              template<typename v1, typename v2>
00813
              struct sub {
                  using type = val<v1::v - v2::v>;
00814
00815
00816
00817
              template<typename v1, typename v2>
00818
              struct mul {
                  using type = val<v1::v* v2::v>;
00819
00820
00821
00822
              template<typename v1, typename v2>
00823
              struct div {
                  using type = val<v1::v / v2::v>;
00824
00825
00826
00827
              template<typename v1, typename v2>
00828
              struct remainder {
00829
                  using type = val<v1::v% v2::v>;
00830
00831
00832
              template<tvpename v1, tvpename v2>
```

```
struct qt {
00834
                  using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
00835
00836
00837
              template<typename v1, typename v2>
00838
              struct 1t {
                  using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
00840
00841
00842
              template<typename v1, typename v2>
00843
              struct eq {
                 using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
00844
00845
00846
00847
              template<typename v>
00848
              struct pos {
00849
                  using type = std::bool_constant<(v::v > 0)>;
00850
00851
00852
           public:
00857
              template<typename v1, typename v2>
00858
              using add_t = typename add<v1, v2>::type;
00859
00864
              template<typename v1, typename v2>
00865
              using sub_t = typename sub<v1, v2>::type;
00866
00871
              template<typename v1, typename v2>
00872
              using mul_t = typename mul<v1, v2>::type;
00873
00879
              template<typename v1, typename v2>
00880
              using div t = typename div<v1, v2>::type;
00881
00886
              template<typename v1, typename v2>
00887
              using mod_t = typename remainder<v1, v2>::type;
00888
00894
              template<typename v1, typename v2>
00895
              using gt_t = typename gt<v1, v2>::type;
00896
00901
              template<typename v1, typename v2>
00902
              static constexpr bool gt_v = gt_t<v1, v2>::value;
00903
00909
              template<typename v1, typename v2>
00910
              using lt_t = typename lt<v1, v2>::type;
00911
              template<typename v1, typename v2> static constexpr bool lt_v = lt_t < v1, v2>::value;
00917
00918
00919
00925
              template<typename v1, typename v2>
00926
              using eq_t = typename eq<v1, v2>::type;
00927
00933
              template<typename v1, typename v2>
00934
              static constexpr bool eq_v = eq_t<v1, v2>::value;
00935
00941
              template<typename v1, typename v2>
00942
              using gcd_t = gcd_t < i64, v1, v2>;
00943
00948
              template<typename v>
00949
              using pos_t = typename pos<v>::type;
00950
00955
              template<typename v>
00956
              static constexpr bool pos_v = pos_t < v > :: value;
00957
          };
00958 } // namespace aerobus
00959
00960 // z/pz
00961 namespace aerobus {
00966
          template<int32_t p>
00967
          struct zpz {
              using inner_type = int32_t;
00968
              template<int32_t x>
00969
00970
              struct val {
                  using enclosing_type = zpz;
static constexpr int32_t v = x % p;
00972
00974
00975
00976
                  template<typename valueType>
00977
                  static constexpr valueType get() { return static_cast<valueType>(x % p); }
00978
00979
                  using is_zero_t = std::bool_constant<x% p == 0>;
00980
                  static std::string to_string() {
00981
                       return std::to string(x % p);
00982
00983
00984
                  template<typename valueRing>
00985
                   static constexpr valueRing eval(const valueRing& v) {
00986
                       return static_cast<valueRing>(x % p);
00987
                   }
00988
              };
```

```
00989
00990
              template<auto x>
00991
              using inject_constant_t = val<static_cast<int32_t>(x)>;
00992
00993
              using zero = val<0>:
00994
              using one = val<1>;
              static constexpr bool is_field = is_prime::value;
00995
00996
              static constexpr bool is_euclidean_domain = true;
00997
00998
           private:
              template<typename v1, typename v2>
00999
01000
              struct add {
01001
                  using type = val<(v1::v + v2::v) % p>;
01002
01003
01004
              template<typename v1, typename v2>
01005
              struct sub {
                 using type = val<(v1::v - v2::v) % p>;
01006
01007
01008
01009
              template<typename v1, typename v2>
01010
              struct mul {
                 using type = val<(v1::v* v2::v) % p>;
01011
01012
01013
01014
              template<typename v1, typename v2>
01015
              struct div
01016
                  using type = val<(v1::v% p) / (v2::v % p)>;
01017
01018
01019
              template<typename v1, typename v2>
01020
              struct remainder {
01021
                 using type = val<(v1::v% v2::v) % p>;
01022
              };
01023
01024
              template<typename v1, typename v2>
01025
              struct qt {
                  using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
01027
01028
01029
              template<typename v1, typename v2>
01030
              struct lt {
                 using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
01031
01032
01033
01034
              template<typename v1, typename v2>
01035
              struct eq {
                  using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
01036
01037
01038
              template<typename v1>
01040
              struct pos {
01041
                  using type = std::bool_constant<(v1::v > 0)>;
01042
01043
01044
           public:
01048
              template<typename v1, typename v2>
              using add_t = typename add<v1, v2>::type;
01049
01050
01054
              template<typename v1, typename v2>
01055
              using sub_t = typename sub<v1, v2>::type;
01056
01060
              template<typename v1, typename v2>
01061
              using mul_t = typename mul<v1, v2>::type;
01062
01066
              template<typename v1, typename v2>
01067
              using div_t = typename div<v1, v2>::type;
01068
              template<typename v1, typename v2>
01072
01073
              using mod_t = typename remainder<v1, v2>::type;
01074
01078
              template<typename v1, typename v2>
01079
              using gt_t = typename gt<v1, v2>::type;
01080
01084
              template<typename v1, typename v2>
01085
              static constexpr bool gt_v = gt_t<v1, v2>::value;
01086
01090
              template<typename v1, typename v2>
01091
              using lt_t = typename lt<v1, v2>::type;
01092
              template<typename v1, typename v2>
static constexpr bool lt_v = lt_t<v1, v2>::value;
01096
01097
01098
01102
              template<typename v1, typename v2>
01103
              using eq_t = typename eq<v1, v2>::type;
01104
01108
              template<tvpename v1, tvpename v2>
```

```
static constexpr bool eq_v = eq_t<v1, v2>::value;
01110
01114
              template<typename v1, typename v2>
01115
             using gcd_t = gcd_t < i32, v1, v2>;
01116
01119
              template<typename v1>
01120
             using pos_t = typename pos<v1>::type;
01121
01124
              template<typename v>
01125
             static constexpr bool pos_v = pos_t<v>::value;
01126
          };
01127 } // namespace aerobus
01128
01129 // polynomial
01130 namespace aerobus {
         // coeffN x^N + ..
01131
01136
         template<typename Ring>
01137
         requires IsEuclideanDomain<Ring>
01138
         struct polynomial {
01139
             static constexpr bool is_field = false;
              static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
01140
01141
01145
              template<typename coeffN, typename... coeffs>
01146
              struct val {
01148
                 using enclosing_type = polynomial<Ring>;
01150
                  static constexpr size_t degree = sizeof...(coeffs);
01152
                  using aN = coeffN;
01154
                  using strip = val<coeffs...>;
                  using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
01156
01158
                  static constexpr bool is_zero_v = is_zero_t::value;
01159
01160
              private:
01161
                 template<size_t index, typename E = void>
01162
                  struct coeff_at {};
01163
01164
                  template<size t index>
                 struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {
01165
                      using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
01166
01167
01168
01169
                  template<size_t index>
                  struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
01170
01171
                     using type = typename Ring::zero;
01172
                  };
01173
01174
               public:
01177
                  template<size_t index>
01178
                  using coeff_at_t = typename coeff_at<index>::type;
01179
01182
                  static std::string to string() {
01183
                     return string_helper<coeffN, coeffs...>::func();
01184
01185
01190
                 template<typename valueRing>
                  static constexpr valueRing eval(const valueRing& x) {
01191
                     return horner_evaluation<valueRing, val>
01192
                             ::template inner<0, degree + 1>
01193
01194
                              ::func(static_cast<valueRing>(0), x);
01195
01196
             };
01197
              template<typename coeffN>
01200
01201
              struct val<coeffN> {
01203
                using enclosing_type = polynomial<Ring>;
01205
                  static constexpr size_t degree = 0;
01206
                  using aN = coeffN;
01207
                 using strip = val<coeffN>;
                 using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
01208
01209
                  static constexpr bool is_zero_v = is_zero_t::value;
01211
01212
                  template<size_t index, typename E = void>
                 struct coeff_at {};
01213
01214
01215
                  template<size t index>
01216
                  struct coeff_at<index, std::enable_if_t<(index == 0)» {</pre>
01217
                      using type = aN;
01218
01219
01220
                  template<size t index>
                  struct coeff at<index, std::enable if t<(index < 0 || index > 0)» {
01221
01222
                      using type = typename Ring::zero;
01223
01224
01225
                  template<size_t index>
01226
                  using coeff_at_t = typename coeff_at<index>::type;
01227
```

```
static std::string to_string() {
01229
                     return string_helper<coeffN>::func();
01230
                  }
01231
01232
                  template<typename valueRing>
                  static constexpr valueRing eval(const valueRing& x) {
01233
                      return static_cast<valueRing>(aN::template get<valueRing>());
01234
01235
01236
              };
01237
01239
              using zero = val<typename Ring::zero>;
              using one = val<typename Ring::one>;
01241
01243
              using X = val<typename Ring::one, typename Ring::zero>;
01244
01245
          private:
01246
              template<typename P, typename E = void>
01247
              struct simplify;
01248
              template <typename P1, typename P2, typename I>
01250
              struct add_low;
01251
01252
              template<typename P1, typename P2>
01253
              struct add {
                  using type = typename simplify<typename add_low<
01254
01255
                  P1,
01256
01257
                  internal::make_index_sequence_reverse<</pre>
01258
                  std::max(P1::degree, P2::degree) + 1
01259
                  »::type>::type;
01260
              };
01261
01262
              template <typename P1, typename P2, typename I>
01263
              struct sub_low;
01264
01265
              template <typename P1, typename P2, typename I>
01266
              struct mul_low;
01267
01268
              template<typename v1, typename v2>
01269
              struct mul {
01270
                      using type = typename mul_low<
01271
01272
                          v2,
01273
                          internal::make_index_sequence_reverse<</pre>
01274
                          v1::degree + v2::degree + 1
01275
                          »::type;
01276
01277
01278
              template<typename coeff, size_t deg>
01279
              struct monomial:
01280
              template<typename v, typename E = void>
01282
              struct derive_helper {};
01283
01284
              template<typename v>
01285
              struct derive_helper<v, std::enable_if_t<v::degree == 0» {</pre>
01286
                  using type = zero;
01287
01288
01289
              template<typename v>
01290
              struct derive_helper<v, std::enable_if_t<v::degree != 0» {</pre>
01291
                  using type = typename add<
01292
                      typename derive_helper<typename simplify<typename v::strip>::type>::type,
01293
                      typename monomial<
01294
                           typename Ring::template mul_t<
01295
                               typename v::aN,
01296
                              typename Ring::template inject_constant_t<(v::degree)>
01297
01298
                          v::dearee - 1
01299
                      >::type
01300
                  >::type;
01301
01302
01303
              template<typename v1, typename v2, typename E = void>
01304
              struct eq_helper {};
01305
01306
              template<typename v1, typename v2>
01307
              struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
01308
                  using type = std::false_type;
01309
              };
01310
01311
01312
              template<typename v1, typename v2>
01313
              struct eq_helper<v1, v2, std::enable_if_t<
01314
                  v1::degree == v2::degree &&
01315
                  (v1::degree != 0 || v2::degree != 0) &&
01316
                  std::is same<
01317
                  typename Ring::template eg t<typename v1::aN, typename v2::aN>,
```

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

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

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

```
01576
                           }
01577
01578
                           if (sizeof...(coeffs) == 1) {
                               result += coeff::to_string() + " x";
01579
01580
                           } else {
01581
                               result += coeff::to_string()
                                        + " x^" + std::to_string(sizeof...(coeffs));
01582
01583
01584
                       }
01585
                       if (!tail.empty()) {
    result += " + " + tail;
01586
01587
01588
01589
01590
                       return result;
01591
                  }
              };
01592
01593
01594
              template<typename coeff>
01595
              struct string_helper<coeff> {
01596
                  static std::string func()
01597
                       if (!std::is_same<coeff, typename Ring::zero>::value) {
01598
                           return coeff::to_string();
01599
                       } else {
01600
                           return "";
01601
01602
                   }
01603
              };
01604
01605
           public:
01608
              template<tvpename P>
01609
              using simplify_t = typename simplify<P>::type;
01610
01614
              template<typename v1, typename v2>
01615
              using add_t = typename add<v1, v2>::type;
01616
              template<typename v1, typename v2>
using sub_t = typename sub<v1, v2>::type;
01620
01621
01622
01626
              template<typename v1, typename v2>
01627
              using mul_t = typename mul<v1, v2>::type;
01628
              template<typename v1, typename v2>
using eq_t = typename eq_helper<v1, v2>::type;
01632
01633
01634
01638
               template<typename v1, typename v2>
01639
              using lt_t = typename lt_helper<v1, v2>::type;
01640
              template<typename v1, typename v2>
01644
01645
              using gt_t = typename gt_helper<v1, v2>::type;
01646
01650
               template<typename v1, typename v2>
01651
              using div_t = typename div<v1, v2>::q_type;
01652
01656
              template<typename v1, typename v2>
01657
              using mod t = typename div helper<v1, v2, zero, v1>::mod type;
01658
01662
               template<typename coeff, size_t deg>
01663
              using monomial_t = typename monomial<coeff, deg>::type;
01664
01667
              template<typename v>
01668
              using derive_t = typename derive_helper<v>::type;
01669
01672
              template<typename v>
01673
              using pos_t = typename Ring::template pos_t<typename v::aN>;
01674
01677
              template<typename v>
01678
              static constexpr bool pos_v = pos_t<v>::value;
01679
              template<typename v1, typename v2>
01684
              using gcd_t = std::conditional_t<
01685
                   Ring::is_euclidean_domain,
01686
                   typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
01687
01688
01692
              template<auto x>
01693
              using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
01694
01698
              template<typename v>
01699
              using inject_ring_t = val<v>;
01700
          };
01701 } // namespace aerobus
01702
01703 // fraction field
01704 namespace aerobus {
01705
          namespace internal {
              template<typename Ring, typename E = void>
01706
```

```
requires IsEuclideanDomain<Ring>
              struct _FractionField {};
01708
01709
01710
              template<typename Ring>
01711
              requires IsEuclideanDomain<Ring>
              struct _FractionFielddRing, std::enable_if_t<Ring::is_euclidean_domain</pre> {
    static constexpr bool is_field = true;
01712
01714
01715
                  static constexpr bool is_euclidean_domain = true;
01716
               private:
01717
01718
                  template<typename val1, typename val2, typename E = void>
01719
                  struct to_string_helper {};
01720
01721
                   template<typename val1, typename val2>
01722
                   struct to_string_helper <val1, val2,
01723
                       std::enable_if_t<
01724
                       Ring::template eq_t<
01725
                       val2, typename Ring::one
                      >::value
01726
01727
01728
01729
                       static std::string func() {
01730
                          return vall::to_string();
01731
01732
                  };
01733
01734
                  template<typename val1, typename val2>
01735
                  struct to_string_helper<val1, val2,
01736
                       std::enable_if_t<
01737
                       !Ring::template eq_t<
01738
                       val2.
01739
                       typename Ring::one
01740
                       >::value
01741
01742
                  > {
01743
                      static std::string func() {
01744
                          return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
01745
01746
                  };
01747
01748
               public:
01752
                  template<typename val1, typename val2>
01753
                  struct val {
01755
                      using x = val1;
01757
                      using y = val2;
                       using is_zero_t = typename vall::is_zero_t;
01759
01761
                      static constexpr bool is_zero_v = val1::is_zero_t::value;
01762
01764
                      using ring_type = Ring;
                      using enclosing_type = _FractionField<Ring>;
01765
01766
01769
                       static constexpr bool is_integer = std::is_same_v<val2, typename Ring::one>;
01770
01774
                      template<typename valueType>
                       static constexpr valueType get() { return static_cast<valueType>(x::v) /
01775
     static cast<valueType>(v::v); }
01776
01779
                       static std::string to_string() {
01780
                           return to_string_helper<val1, val2>::func();
01781
01782
01787
                      template<typename valueRing>
01788
                      static constexpr valueRing eval(const valueRing& v) {
01789
                         return x::eval(v) / y::eval(v);
01790
01791
                  };
01792
01794
                  using zero = val<tvpename Ring::zero, tvpename Ring::one>;
01796
                  using one = val<typename Ring::one, typename Ring::one>;
01797
01800
                  template<typename v>
01801
                  using inject_t = val<v, typename Ring::one>;
01802
01805
                  template<auto x>
                  using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
01806
     Ring::one>;
01807
01810
                  template<typename v>
01811
                  using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
01812
01814
                  using ring_type = Ring;
01815
01816
01817
                  template<typename v, typename E = void>
01818
                  struct simplify {};
01819
01820
                  // x = 0
```

```
template<typename v>
                  struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {</pre>
01822
01823
                       using type = typename _FractionField<Ring>::zero;
01824
01825
01826
                   // x != 0
                  template<typename v>
01827
01828
                   struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {
01829
01830
                       using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
                       using newx = typename Ring::template div_t<typename v::x, _gcd>;
01831
01832
                      using newy = typename Ring::template div_t<typename v::y, _gcd>;
01833
01834
                       using posx = std::conditional_t<
01835
                                            !Ring::template pos_v<newy>,
01836
                                            typename Ring::template sub_t<typename Ring::zero, newx>,
01837
                                            newx>:
01838
                      using posy = std::conditional_t<
01839
                                            !Ring::template pos_v<newy>,
01840
                                            typename Ring::template sub_t<typename Ring::zero, newy>,
01841
                   public:
01842
01843
                       using type = typename _FractionField<Ring>::template val<posx, posy>;
01844
                  }:
01845
01846
               public:
01849
                  template<typename v>
01850
                  using simplify_t = typename simplify<v>::type;
01851
01852
               private:
01853
                  template<tvpename v1, tvpename v2>
01854
                  struct add {
01855
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01856
01857
01858
                       using dividend = typename Ring::template add_t<a, b>;
01859
                      using diviser = typename Ring::template mul t<typename v1::y, typename v2::y>;
                      using g = typename Ring::template gcd_t<dividend, diviser>;
01860
01861
01862
                   public:
01863
                       using type = typename _FractionField<Ring>::template simplify_t<val<dividend,
     diviser»:
01864
                   }:
01865
                   template<typename v>
01866
01867
                   struct pos {
01868
                       using type = std::conditional_t<
                           (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
01869
01870
                            (!Ring::template pos_v < typename v::x > \&\& !Ring::template pos_v < typename v::y >) \textit{,} \\
01871
                           std::true type,
01872
                           std::false_type>;
01873
01874
01875
                  template<typename v1, typename v2>
01876
                  struct sub {
01877
                   private:
01878
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
01879
                       using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01880
                       using dividend = typename Ring::template sub_t<a, b>;
01881
                       using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01882
                      using g = typename Ring::template gcd_t<dividend, diviser>;
01883
01884
                   public:
                      using type = typename _FractionField<Ring>::template simplify_t<val<dividend,</pre>
     diviser»;
01886
01887
01888
                  template<typename v1, typename v2>
01889
                   struct mul {
01890
                   private:
01891
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
01892
                       using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
01893
01894
                   public:
01895
                      using type = typename FractionField<Ring>::template simplify t<val<a, b>;
01896
                  };
01897
01898
                  template<typename v1, typename v2, typename E = void>
01899
                   struct div {};
01900
                  template<typename v1, typename v2>
01901
                   struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
01902
      _FractionField<Ring>::zero>::value» {
                   private:
01903
01904
                      using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
                       using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
01905
01906
```

```
public:
                      using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;</pre>
01908
01909
                  };
01910
01911
                  template<typename v1, typename v2>
                  struct div<v1, v2, std::enable_if_t<
01912
                      std::is_same<zero, v1>::value && std::is_same<v2, zero>::value» {
01913
01914
                      using type = one;
01915
                  };
01916
                  template<typename v1, typename v2> ^{\circ}
01917
01918
                  struct eq {
01919
                      using type = std::conditional_t<
01920
                              ...
std::is_same<typename simplify_t<vl>::x, typename simplify_t<v2>::x>::value &&
01921
                               std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
01922
                           std::true_type,
01923
                           std::false_type>;
01924
                  };
01925
01926
                  template<typename v1, typename v2, typename E = void>
01927
01928
01929
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
01930
                      (eq<v1, v2>::type::value)
01931
01932
                      using type = std::false_type;
01933
01934
01935
                  template<typename v1, typename v2>
01936
                  struct gt<v1, v2, std::enable_if_t<
(!eq<v1, v2>::type::value) &&
01937
01938
01939
                       (!pos<v1>::type::value) && (!pos<v2>::type::value)
01940
01941
                      using type = typename gt<
01942
                          typename sub<zero, v1>::type, typename sub<zero, v2>::type
01943
                      >::type;
01944
                  } ;
01945
01946
                  template<typename v1, typename v2>
01947
                  struct gt<v1, v2, std::enable_if_t<
                      (!eq<v1, v2>::type::value) &&
01948
01949
                       (pos<v1>::type::value) && (!pos<v2>::type::value)
01950
01951
                      using type = std::true_type;
01952
                  };
01953
                  01954
01955
01956
01957
                       (!pos<v1>::type::value) && (pos<v2>::type::value)
01958
01959
                      using type = std::false_type;
01960
                  };
01961
01962
                  template<typename v1, typename v2>
                  struct gt<v1, v2, std::enable_if_t<
01963
01964
                       (!eq<v1, v2>::type::value) &&
01965
                       (pos<v1>::type::value) && (pos<v2>::type::value)
01966
01967
                      using type = typename Ring::template gt t<
01968
                          typename Ring::template mul_t<v1::x, v2::y>,
01969
                           typename Ring::template mul_t<v2::y, v2::x>
01970
01971
                  };
01972
01973
               public:
01978
                  template<typename v1, typename v2>
01979
                  using add t = typename add<v1, v2>::type;
01980
01985
                  template<typename v1, typename v2>
01986
                  using mod_t = zero;
01987
01992
                  template<typename v1, typename v2>
01993
                  using gcd_t = v1;
01994
01998
                  template<typename v1, typename v2>
01999
                  using sub_t = typename sub<v1, v2>::type;
02000
02004
                  template<typename v1, typename v2>
02005
                  using mul_t = typename mul<v1, v2>::type;
02006
02010
                  template<typename v1, typename v2>
02011
                  using div_t = typename div<v1, v2>::type;
02012
02016
                  template<typename v1, typename v2>
02017
                  using eq_t = typename eq<v1, v2>::type;
```

```
02018
02022
                  template<typename v1, typename v2>
02023
                   static constexpr bool eq_v = eq<v1, v2>::type::value;
02024
02028
                  template<typename v1, typename v2>
02029
                  using gt t = typename gt<v1, v2>::type;
02030
02034
                  template<typename v1, typename v2>
02035
                  static constexpr bool gt_v = gt<v1, v2>::type::value;
02036
02039
                  template<typename v1>
02040
                  using pos_t = typename pos<v1>::type;
02041
02044
                  template<typename v>
02045
                  static constexpr bool pos_v = pos_t<v>::value;
02046
              };
02047
02048
              template<typename Ring, typename E = void>
              requires IsEuclideanDomain<Ring>
02049
02050
              struct FractionFieldImpl {};
02051
02052
              // fraction field of a field is the field itself
02053
              template<typename Field>
              requires IsEuclideanDomain<Field>
02054
02055
              struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {</pre>
02056
                using type = Field;
02057
                  template<typename v>
02058
                  using inject_t = v;
02059
              };
02060
02061
              \ensuremath{//} fraction field of a ring is the actual fraction field
02062
              template<typename Ring>
02063
              requires IsEuclideanDomain<Ring>
02064
              struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {</pre>
02065
                 using type = _FractionField<Ring>;
02066
              };
          } // namespace internal
02067
02068
02072
          template<typename Ring>
02073
          requires IsEuclideanDomain<Ring>
02074
          using FractionField = typename internal::FractionFieldImpl<Ring>::type;
02075 } // namespace aerobus
02076
02077 // short names for common types
02078 namespace aerobus {
02081
          using q32 = FractionField<i32>;
02084
          using fpq32 = FractionField<polynomial<q32>>;
02087
          using q64 = FractionField<i64>;
          using pi64 = polynomial<i64>;
02089
          using pq64 = polynomial<q64>;
using pq64 = FractionField<polynomial<q64>>;
02091
02093
02098
          template<typename Ring, typename v1, typename v2>
02099
          using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
02100
          template<int64_t p, int64_t q>
using make_q64_t = typename q64::template simplify_t<</pre>
02104
02105
02106
                       typename q64::val<i64::inject_constant_t<p>, i64::inject_constant_t<q>»;
02107
          template<int32_t p, int32_t q>
using make_q32_t = typename q32::template simplify_t<</pre>
02111
02112
                       typename q32::val<i32::inject_constant_t<p>, i32::inject_constant_t<q>»;
02113
02114
02119
          template<typename Ring, typename v1, typename v2>
02120
          using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
02125
          template<typename Ring, typename v1, typename v2>
02126
          using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
02127 }
        // namespace aerobus
02128
02129 // taylor series and common integers (factorial, bernoulli...) appearing in taylor coefficients
02130 namespace aerobus {
02131
       namespace internal {
02132
             template<typename T, size_t x, typename E = void>
02133
             struct factorial { };
02134
02135
              template<typename T, size t x>
02136
              struct factorial<T, x, std::enable_if_t<(x > 0)» {
02137
              private:
02138
                 template<typename, size_t, typename>
02139
                  friend struct factorial;
              public:
02140
                  using type = typename T::template mul t<typename T::template val<x>, typename factorial<T,
02141
     x - 1>::type>;
02142
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02143
             };
02144
02145
              template<tvpename T>
```

```
02146
              struct factorial<T, 0> {
02147
               public:
02148
                  using type = typename T::one;
02149
                  static constexpr typename T::inner_type value = type::template get<typename</pre>
     T::inner_type>();
02150
              };
          } // namespace internal
02151
02152
02156
          template<typename T, size_t i>
02157
          using factorial_t = typename internal::factorial<T, i>::type;
02158
02162
          template<typename T, size_t i>
          inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
02163
02164
02165
02166
              template<typename T, size_t k, size_t n, typename E = void>
02167
              struct combination_helper {};
02168
02169
              template<typename T, size_t k, size_t n>
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» { using type = typename FractionField<T>::template mul_t<
02170
02171
                       typename combination_helper<T, k - 1, n - 1>::type,
02172
                       makefraction_t<T, typename T::template val<n>, typename T::template val<k>>;
02173
02174
              };
02175
02176
              template<typename T, size_t k, size_t n>
02177
              struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)   {
02178
                  using type = typename combination_helper<T, n - k, n>::type;
02179
02180
02181
              template<typename T, size t n>
02182
              struct combination_helper<T, 0, n> {
02183
                  using type = typename FractionField<T>::one;
02184
              };
02185
              template<typename T, size_t k, size_t n>
02186
02187
              struct combination {
02188
                  using type = typename internal::combination_helper<T, k, n>::type::x;
02189
                  static constexpr typename T::inner_type value =
02190
                               internal::combination_helper<T, k, n>::type::template get<typename</pre>
     T::inner_type>();
02191
              };
          } // namespace internal
02192
02193
          template<typename T, size_t k, size_t n>
02196
02197
          using combination_t = typename internal::combination<T, k, n>::type;
02198
          template<typename T, size_t k, size_t n>
inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
02203
02204
02205
02206
          namespace internal {
02207
              template<typename T, size_t m>
02208
              struct bernoulli;
02209
02210
              template<typename T, typename accum, size_t k, size_t m>
02211
              struct bernoulli_helper {
02212
                  using type = typename bernoulli_helper<
02213
02214
                       addfractions_t<T,
02215
                           accum,
                           mulfractions_t<T,
02216
02217
                               makefraction t<T,
02218
                                    combination_t<T, k, m + 1>,
02219
                                    typename T::one>,
02220
                                typename bernoulli<T, k>::type
02221
                           >
02222
                       >,
k + 1,
02223
02224
                       m>::tvpe;
02225
              };
02226
02227
              template<typename T, typename accum, size_t m>
02228
              struct bernoulli_helper<T, accum, m, m> {
02229
                  using type = accum;
02230
02231
02232
02233
02234
               template<typename T, size_t m>
02235
               struct bernoulli {
                  using type = typename FractionField<T>::template mul_t<</pre>
02236
02237
                       typename internal::bernoulli_helper<T, typename FractionField<T>::zero, 0, m>::type,
02238
                       makefraction_t<T,</pre>
02239
                       typename T::template val<static_cast<typename T::inner_type>(-1)>,
02240
                       typename T::template val<static_cast<typename T::inner_type>(m + 1)>
02241
02242
                   >;
```

```
02243
                   template<typename floatType>
02244
02245
                   static constexpr floatType value = type::template get<floatType>();
02246
              };
02247
02248
              template<typename T>
02249
              struct bernoulli<T, 0> {
02250
                   using type = typename FractionField<T>::one;
02251
02252
                  template<typename floatType>
02253
                  static constexpr floatType value = type::template get<floatType>();
02254
              };
02255
          } // namespace internal
02256
02260
          template<typename T, size_t n>
02261
          using bernoulli_t = typename internal::bernoulli<T, n>::type;
02262
          template<typename FloatType, typename T, size_t n >
inline constexpr FloatType bernoulli_v = internal::bernoulli<T, n>::template value<FloatType>;
02267
02268
02269
02270
          namespace internal {
              template<typename T, int k, typename E = void>
02271
              struct alternate {};
02272
02273
02274
              template<typename T, int k>
02275
              struct alternate<T, k, std::enable_if_t<k % 2 == 0» {</pre>
02276
                   using type = typename T::one;
02277
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02278
             };
02279
02280
              template<typename T, int k>
02281
              struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
02282
                   using type = typename T::template sub_t<typename T::zero, typename T::one>;
02283
                  static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
02284
              };
          } // namespace internal
02285
02286
02289
          template<typename T, int k>
02290
          using alternate_t = typename internal::alternate<T, k>::type;
02291
02292
          namespace internal {
02293
              template<typename T, int n, int k, typename E = void>
02294
              struct stirling_helper {};
02295
02296
              template<typename T>
              struct stirling_helper<T, 0, 0> {
02297
02298
                  using type = typename T::one;
02299
02300
02301
              template<typename T, int n>
02302
              struct stirling_helper<T, n, 0, std::enable_if_t<(n > 0)» {
02303
                  using type = typename T::zero;
02304
02305
02306
              template<typename T, int n>
              struct stirling_helper<T, 0, n, std::enable_if_t<(n > 0)» {
02307
02308
                   using type = typename T::zero;
02309
02310
              template<typename T, int n, int k >
02311
02312
              struct stirling_helper<T, n, k, std::enable_if_t<(k > 0) && (n > 0)» {
02313
                  using type = typename T::template sub_t<
02314
                                    typename stirling_helper<T, n-1, k-1>::type,
02315
                                    typename T::template mul_t<</pre>
02316
                                        typename T::template inject_constant_t<n-1>,
                                        typename stirling_helper<T, n-1, k>::type
02317
02318
02319
02320
          } // namespace internal
02321
02326
          template<typename T, int n, int k>
          using stirling_signed_t = typename internal::stirling_helper<T, n, k>::type;
02327
02328
02333
          template<typename T, int n, int k>
02334
          using stirling_unsigned_t = abs_t<typename internal::stirling_helper<T, n, k>::type>;
02335
          template<typename T, int n, int k>
static constexpr typename T::inner_type stirling_signed_v = stirling_signed_t<T, n, k>::v;
02340
02341
02342
02343
02348
          template<typename T, int n, int k>
02349
          static constexpr typename T::inner_type stirling_unsigned_v = stirling_unsigned_t<T, n, k>::v;
02350
          template<typename T, size_t k>
inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
02353
02354
```

```
02355
02356
          namespace internal {
02357
               template<typename T, auto p, auto n, typename E = void>
02358
               struct pow {};
02359
               template<typename T, auto p, auto n>
02360
               struct pow<T, p, n, std::enable_if_t<(n > 0 && n % 2 == 0)» {
02361
02362
                   using type = typename T::template mul_t<</pre>
02363
                      typename pow<T, p, n/2>::type,
02364
                        typename pow<T, p, n/2>::type
02365
                   >;
02366
               };
02367
02368
               template<typename T, auto p, auto n>
02369
               struct pow<T, p, n, std::enable_if_t<(n % 2 == 1)» {
                   using type = typename T::template mul_t<
    typename T::template inject_constant_t<p>,
02370
02371
02372
                        typename T::template mul_t<
                            typename pow<T, p, n/2>::type, typename pow<T, p, n/2>::type
02374
02375
02376
                   >;
02377
               };
02378
02379
               template<typename T, auto n, auto p>
               struct pow<T, n, p, std::enable_if_t<p == 0» { using type = typename T::one; };
02380
02381
             // namespace internal
02382
02387
          template<typename T, auto p, auto n>
02388
          using pow_t = typename internal::pow<T, p, n>::type;
02389
02394
           template<typename T, auto p, auto n>
02395
          static constexpr typename T::inner_type pow_v = internal::pow<T, p, n>::type::v;
02396
          namespace internal {
02397
               template<typename, template<typename, size_t> typename, class>
02398
02399
               struct make_taylor_impl;
02401
               template<typename T, template<typename, size_t> typename coeff_at, size_t... Is>
02402
               struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {</pre>
02403
                   using type = typename polynomial<FractionField<T>>::template val<typename coeff_at<T,
      Is>::type...>;
02404
              };
02405
02406
           template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
02411
02412
          using taylor = typename internal::make_taylor_impl<</pre>
02413
02414
               coeff at.
02415
               internal::make index sequence reverse<deg + 1>>::type;
02416
02417
          namespace internal {
02418
               template<typename T, size_t i>
02419
               struct exp_coeff {
                   using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02420
02421
               };
02422
02423
               template<typename T, size_t i, typename E = void>
02424
               struct sin_coeff_helper {};
02425
02426
               template<typename T, size_t i>
               struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
02427
02428
02429
02430
02431
               template<typename T, size_t i>
               struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
    using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
02432
02433
02434
02435
02436
               template<typename T, size_t i>
02437
               struct sin_coeff {
02438
                   using type = typename sin_coeff_helper<T, i>::type;
02439
02440
02441
               template<typename T, size_t i, typename E = void>
02442
               struct sh_coeff_helper {};
02443
02444
               template<typename T, size_t i>
               struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
    using type = typename FractionField<T>::zero;
02445
02446
02447
               };
02448
02449
               template<typename T, size_t i>
02450
               struct sh\_coeff\_helper<T, i, std::enable\_if\_t<(i & 1) == 1» {
02451
                   using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
02452
               };
```

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

```
02540
              };
02541
02542
              template<typename T, size_t i>
02543
              struct lnp1_coeff {
02544
                  using type = makefraction_t<T,
02545
                       alternate_t<T, i + 1>,
02546
                       typename T::template val<i>;;
02547
02548
02549
              template<typename T>
              struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
02550
02551
02552
              template<typename T, size_t i, typename E = void>
02553
              struct asinh_coeff_helper;
02554
              template<typename T, size_t i>
struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {</pre>
02555
02556
                  using type = makefraction_t<T,
02557
                       typename T::template mul_t<
02558
02559
                           alternate_t<T, i / 2>,
02560
                           factorial_t<T, i - 1>
02561
02562
                       typename T::template mul_t<
02563
                           typename T::template mul_t<</pre>
02564
                               typename T::template val<i>,
02565
                               pow_t<T, (factorial<T, i / 2>::value), 2>
02566
02567
                           pow_t<T, 4, i / 2>
02568
02569
                  >;
02570
              };
02572
              template<typename T, size_t i>
02573
               struct asinh\_coeff\_helper<T, i, std::enable\_if\_t<(i & 1) == 0» {
02574
                  using type = typename FractionField<T>::zero;
02575
02576
02577
              template<typename T, size_t i>
02578
              struct asinh_coeff {
02579
                 using type = typename asinh_coeff_helper<T, i>::type;
02580
02581
              template<typename T, size_t i, typename E = void>
02582
02583
              struct atanh_coeff_helper;
02584
02585
               template<typename T, size_t i>
02586
               struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
02587
                  // 1/i
02588
                  using type = typename FractionField<T>:: template val<
02589
                      typename T::one,
02590
                       typename T::template inject_constant_t<i>;;
02591
02592
              template<typename T, size_t i>
struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {</pre>
02593
02594
                  using type = typename FractionField<T>::zero;
02595
02596
02597
02598
              template<typename T, size_t i>
02599
              struct atanh_coeff {
                  using type = typename atanh_coeff_helper<T, i>::type;
02600
02601
02602
02603
              template<typename T, size_t i, typename E = void>
02604
              struct tan_coeff_helper;
02605
02606
              template<typename T, size_t i>
              struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {</pre>
02607
                  using type = typename FractionField<T>::zero;
02608
02610
02611
              template<typename T, size_t i>
02612
              struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02613
              private:
                   // 4^((i+1)/2)
02614
02615
                   using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
02616
                   // 4^((i+1)/2)
02617
                   using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
     FractionField<T>::one>;
02618
                  //(-1)^{(i-1)/2}
02619
                   using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
02620
                   using dividend = typename FractionField<T>::template mul_t<</pre>
02621
                       altp,
02622
                       FractionField<T>::template mul_t<
02623
                       FractionField<T>::template mul t<
02624
02625
                       4pm1,
```

```
bernoulli_t<T, (i + 1)>
02627
02628
02629
                  >:
02630
              public:
                  using type = typename FractionField<T>::template div_t<dividend,</pre>
02631
                      typename FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02632
02633
02634
02635
              template<typename T, size_t i>
02636
              struct tan_coeff {
02637
                 using type = typename tan_coeff_helper<T, i>::type;
02638
02639
02640
              template<typename T, size_t i, typename E = void>
02641
              struct tanh_coeff_helper;
02642
02643
              template<typename T, size t i>
              struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
02644
                  using type = typename FractionField<T>::zero;
02645
02646
02647
              template<typename T, size_t i>
struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {</pre>
02648
02649
02650
              private:
                 using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
02651
                  using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename</pre>
02652
     FractionField<T>::one>;
02653
                  using dividend =
                      typename FractionField<T>::template mul_t<</pre>
02654
02655
                           4p.
02656
                          typename FractionField<T>::template mul_t<</pre>
02657
                              _4pm1,
02658
                               bernoulli_t<T, (i + 1) >>::type;
              public:
02659
               using type = typename FractionField<T>::template div_t<dividend,
02660
                     FractionField<T>::template inject_t<factorial_t<T, i + 1>>;
02661
02662
02663
02664
              template<typename T, size_t i>
              struct tanh_coeff {
02665
                 using type = typename tanh_coeff_helper<T, i>::type;
02666
02667
02668
          } // namespace internal
02669
02673
          template<typename Integers, size_t deg>
02674
          using exp = taylor<Integers, internal::exp_coeff, deg>;
02675
02679
          template<typename Integers, size_t deg>
          using expm1 = typename polynomial<FractionField<Integers>>::template sub_t
02680
02681
              exp<Integers, deg>,
02682
              typename polynomial<FractionField<Integers>>::one>;
02683
02687
          template<typename Integers, size_t deg>
02688
          using lnp1 = taylor<Integers, internal::lnp1_coeff, deg>;
02689
02693
          template<typename Integers, size_t deg>
02694
          using atan = taylor<Integers, internal::atan_coeff, deg>;
02695
02699
          template<typename Integers, size_t deg>
          using sin = taylor<Integers, internal::sin_coeff, deg>;
02700
02701
02705
          template<typename Integers, size_t deg>
02706
          using sinh = taylor<Integers, internal::sh_coeff, deg>;
02707
02712
          template<typename Integers, size_t deg>
02713
          using cosh = taylor<Integers, internal::cosh_coeff, deg>;
02714
          template<typename Integers, size_t deg>
02719
02720
          using cos = taylor<Integers, internal::cos_coeff, deg>;
02721
02726
          template<typename Integers, size_t deg>
02727
          using geometric_sum = taylor<Integers, internal::geom_coeff, deg>;
02728
02733
          template<typename Integers, size t deg>
02734
          using asin = taylor<Integers, internal::asin_coeff, deg>;
02735
02740
          template<typename Integers, size_t deg>
02741
          using asinh = taylor<Integers, internal::asinh_coeff, deg>;
02742
02747
          template<typename Integers, size t deg>
02748
          using atanh = taylor<Integers, internal::atanh_coeff, deg>;
02749
02754
          template<typename Integers, size_t deg>
02755
          using tan = taylor<Integers, internal::tan_coeff, deg>;
02756
02761
          template<typename Integers, size t deg>
```

```
using tanh = taylor<Integers, internal::tanh_coeff, deg>;
02763 }
              // namespace aerobus
02764
02765 // continued fractions
02766 namespace aerobus {
02775
                 template<int64_t... values>
02776
                 struct ContinuedFraction {};
02777
02780
                 template<int64_t a0>
02781
                 struct ContinuedFraction<a0> {
                       using type = typename q64::template inject_constant_t<a0>;
02783
                        static constexpr double val = static_cast<double>(a0);
02785
02786
                 };
02787
02791
                 template<int64_t a0, int64_t... rest>
02792
                 struct ContinuedFraction<a0, rest...> {
02794
                        using type = q64::template add_t<
02795
                                      typename q64::template inject_constant_t<a0>,
                                      typename q64::template div_t<
02796
02797
                                             typename q64::one,
                                             typename ContinuedFraction<rest...>::type
02798
02799
02801
                       static constexpr double val = type::template get<double>();
02802
                 };
02803
02808
                 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>;
02811
                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>;
02813
                using SQRT2_fraction =
         02815
                using SQRT3_fraction
          ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 
          // NOLINT
02816 } // namespace aerobus
02817
02818 // known polynomials
02819 namespace aerobus {
                 // CChebyshev
02820
02821
                 namespace internal {
02822
                        template<int kind, size_t deg>
                        struct chebyshev_helper {
02823
                              using type = typename pi64::template sub_t<
02824
                                      typename pi64::template mul_t<
02825
                                             typename pi64::template mul_t<
02826
02827
                                                   pi64::inject_constant_t<2>,
02828
                                                    typename pi64::X>,
02829
                                             typename chebyshev_helper<kind, deg - 1>::type
02830
02831
                                      typename chebyshev helper<kind, deg - 2>::type
02832
                              >;
02833
                       };
02834
02835
                        template<>
                        struct chebyshev_helper<1, 0> {
02836
02837
                              using type = typename pi64::one;
02838
02839
02840
                        template<>
02841
                        struct chebyshev_helper<1, 1> {
02842
                              using type = typename pi64::X;
02843
02844
02845
02846
                        struct chebyshev_helper<2, 0> {
02847
                              using type = typename pi64::one;
02848
02849
02850
                        template<>
                        struct chebyshev_helper<2, 1> {
02852
                              using type = typename pi64::template mul_t<
02853
                                      typename pi64::inject_constant_t<2>,
02854
                                     typename pi64::X>;
02855
                        };
                 } // namespace internal
02856
02857
02858
                  // Laguerre
02859
                 namespace internal {
02860
                        template<size_t deg>
                        struct laguerre_helper {
02861
02862
                          private:
02863
                               // Lk = (1 / k) * ((2 * k - 1 - x) * 1km1 - (k - 2)Lkm2)
02864
                               using lnm2 = typename laguerre_helper<deg - 2>::type;
02865
                               using lnm1 = typename laguerre_helper<deg - 1>::type;
02866
                               // -x + 2k-1
02867
                               using p = typename pq64::template val <
                                      typename q64::template inject_constant_t<-1>,
02868
```

```
typename q64::template inject_constant_t<2 * deg - 1»;
                  // 1/n
02870
02871
                  using factor = typename pq64::template inject_ring_t<
02872
                      q64::val<typename i64::one, typename i64::template inject_constant_t<deg>>;
02873
02874
               public:
02875
                 using type = typename pq64::template mul_t <</pre>
02876
02877
                      typename pq64::template sub_t<
02878
                         typename pq64::template mul_t<
02879
                             p,
02880
                              lnm1
02881
02882
                          typename pq64::template mul_t<
02883
                              typename pq64::template inject_constant_t<deg-1>,
02884
                              lnm2
02885
02886
02887
                 >;
02888
             } ;
02889
02890
              template<>
02891
              struct laguerre_helper<0> {
02892
                 using type = typename pq64::one;
02893
             };
02894
02895
              template<>
02896
              struct laguerre_helper<1> {
                using type = typename pq64::template sub_t<typename pq64::one, typename pq64::X>;
02897
02898
02899
          } // namespace internal
02900
02901
02902
          namespace internal {
02903
             template<size_t i, size_t m, typename E = void>
02904
             struct bernstein_helper {};
02905
02906
             template<>
02907
             struct bernstein_helper<0, 0> {
02908
                using type = typename pi64::one;
02909
             };
02910
             template<size_t i, size_t m>
02911
             struct bernstein_helper<i, m, std::enable_if_t<
02912
02913
                         (m > 0) && (i == 0)   {
02914
                 using type = typename pi64::mul_t<</pre>
02915
                         typename pi64::sub_t<typename pi64::one, typename pi64::X>,
02916
                         typename bernstein_helper<i, m-1>::type>;
02917
             };
02918
              template<size_t i, size_t m>
             02920
02921
02922
02923
                         typename pi64::X,
                         typename bernstein_helper<i-1, m-1>::type>;
02924
02925
02926
02927
              template<size_t i, size_t m>
             02928
02929
                 using type = typename pi64::add_t<
02930
02931
                         typename pi64::mul_t<
02932
                             typename pi64::sub_t<typename pi64::one, typename pi64::X>,
02933
                              typename bernstein_helper<i, m-1>::type>,
02934
                         typename pi64::mul_t<
02935
                             typename pi64::X,
02936
                             typename bernstein helper<i-1, m-1>::type»;
02937
          };
} // namespace internal
02938
02939
02940
          namespace known_polynomials {
             enum hermite_kind {
    probabilist,
02942
02944
02946
                 physicist
02947
             };
02948
         }
02949
          // hermite
02950
         namespace internal {
02951
02952
             template<size_t deg, known_polynomials::hermite_kind kind>
             struct hermite_helper {};
02954
02955
             template<size_t deg>
02956
             struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist> {
02957
              private:
02958
                 using hnm1 = typename hermite helper<deg - 1.
```

```
known_polynomials::hermite_kind::probabilist>::type;
02959
                  using hnm2 = typename hermite_helper<deg - 2,
      known_polynomials::hermite_kind::probabilist>::type;
02960
02961
               public:
                  using type = typename pi64::template sub_t<
02962
02963
                      typename pi64::template mul_t<typename pi64::X, hnm1>,
02964
                      typename pi64::template mul_t<
02965
                           typename pi64::template inject_constant_t<deg - 1>,
02966
                          hnm2
02967
02968
                  >;
02969
              };
02970
02971
              template<size_t deg>
02972
              struct hermite_helper<deg, known_polynomials::hermite_kind::physicist> {
               private:
02973
02974
                  using hnm1 = typename hermite_helper<deg - 1,
      known_polynomials::hermite_kind::physicist>::type;
02975
                  using hnm2 = typename hermite_helper<deg - 2,
      known_polynomials::hermite_kind::physicist>::type;
02976
02977
               public:
02978
                  using type = typename pi64::template sub_t<</pre>
02979
                       // 2X Hn-1
02980
                       typename pi64::template mul_t<
02981
                           typename pi64::val<typename i64::template inject_constant_t<2>,
02982
                          typename i64::zero>, hnm1>,
02983
02984
                      typename pi64::template mul_t<
02985
                          typename pi64::template inject_constant_t<2*(deg - 1)>,
02986
                          hnm2
02987
02988
02989
              };
02990
02991
              template<>
02992
              struct hermite_helper<0, known_polynomials::hermite_kind::probabilist> {
02993
                  using type = typename pi64::one;
02994
02995
02996
              template<>
02997
              struct hermite helper<1, known polynomials::hermite kind::probabilist> {
02998
                  using type = typename pi64::X;
02999
03000
03001
              template<>
03002
              struct hermite_helper<0, known_polynomials::hermite_kind::physicist> {
03003
                  using type = typename pi64::one;
03004
03005
03006
              template<>
03007
              struct hermite_helper<1, known_polynomials::hermite_kind::physicist> {
03008
                  // 2X
                  using type = typename pi64::template val<typename i64::template inject_constant_t<2>,
03009
      typename i64::zero>;
03010
              };
03011
             // namespace internal
03012
03013
          // legendre
03014
          namespace internal {
03015
              template<size t n>
03016
              struct legendre_helper {
03017
               private:
03018
                  // 1/n constant
                  // (2n-1)/n X
03019
03020
                  using fact_left = typename pq64::monomial_t<make_q64_t<2*n-1, n>, 1>;
03021
                  // (n-1) / n
03022
                  using fact right = typename pg64::val<make g64 t<n-1, n»;
03023
               public:
03024
                  using type = pq64::template sub_t<
03025
                           typename pq64::template mul_t<
                               fact_left,
03026
03027
                               typename legendre_helper<n-1>::type
03028
03029
                           typename pq64::template mul_t<
03030
                               fact_right,
03031
                               typename legendre_helper<n-2>::type
03032
03033
                      >:
03034
              };
03035
03036
              template<>
03037
              struct legendre_helper<0> {
03038
                  using type = typename pq64::one;
03039
              };
03040
```

```
template<>
                               struct legendre_helper<1> {
03042
03043
                                       using type = typename pq64::X;
03044
03045
                      } // namespace internal
03046
03047
                      // bernoulli polynomials
03048
                      namespace internal {
03049
                               template<size_t n>
03050
                               struct bernoulli coeff {
03051
                                       template<typename T, size_t i>
03052
                                        struct inner {
03053
                                         private:
                                                 using F = FractionField<T>;
03054
03055
                                          public:
03056
                                                using type = typename F::template mul_t<
03057
                                                          typename F::template inject_ring_t<combination_t<T, i, n»,
03058
                                                          bernoulli t<T, n-i>
03059
03060
                                       };
03061
03062
                     } // namespace internal
03063
03065
                     namespace known_polynomials {
03072
                               template <size_t deg>
                               using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
03073
03074
03081
                               template <size_t deg>
03082
                               using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
03083
03090
                               template <size t deg>
03091
                               using laguerre = typename internal::laguerre_helper<deg>::type;
03092
03099
                               template <size_t deg>
03100
                               using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist>::type;
03101
03108
                               template <size t deg>
03109
                               using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist>::type;
03110
03118
                               template<size_t i, size_t m>
03119
                               using bernstein = typename internal::bernstein_helper<i, m>::type;
03120
03127
                               template<size t deg>
03128
                               using legendre = typename internal::legendre_helper<deg>::type;
03129
03136
03137
                               using bernoulli = taylor<i64, internal::bernoulli_coeff<deg>::template inner, deg>;
03138
                            // namespace known_polynomials
03139 } // namespace aerobus
03140
03141
03142 #ifdef AEROBUS_CONWAY_IMPORTS
03143
03144 // conway polynomials
03145 namespace aerobus {
                    template<int p, int n>
struct ConwayPolynomial {};
03149
03151
03152 #ifndef DO_NOT_DOCUMENT
03153
                      #define ZPZV ZPZ::template val
                      #define POLYV aerobus::polynomial<ZPZ>::template val
03154
                      template<> struct ConwayPolynomial<2, 1> { using ZPZ = aerobus::zpz<2>; using type =
03155
            POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
                      template<> struct ConwayPolynomial<2, 2> { using ZPZ = aerobus::zpz<2>; using type =
            POLYV<ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
03157
                     template<> struct ConwayPolynomial<2, 3> { using ZPZ = aerobus::zpz<2>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT template<> struct ConwayPolynomial<2, 4> { using ZPZ = aerobus::zpz<2>; using type =
03158
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                      template<> struct ConwayPolynomial<2, 5> { using ZPZ = aerobus::zpz<2>; using type =
03159
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
03160
                    template<> struct ConwayPolynomial<2, 6> { using ZPZ = aerobus::zpz<2>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<2, 7> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>; }; // NOLINT
03161
                      template<> struct ConwayPolynomial<2, 8> { using ZPZ = aerobus::zpz<2>; using type =
03162
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; };
03163
                     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
             NOLINT
             template<> struct ConwayPolynomial<2, 10> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
03164
             ZPZV<1»; }; // NOLINT</pre>
03165
                     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<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1»; }; // NOLINT
                    template<> struct ConwayPolynomial<2, 12> { using ZPZ = aerobus::zpz<2>; using type =
03166
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>,
                                        ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                                              template<> struct ConwayPolynomial<2, 13> { using ZPZ = aerobus::zpz<2>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<2, 14> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , Z
03168
                                         ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                              template<> struct ConwayPolynomial<2, 15> { using ZPZ = aerobus::zpz<2>; using type =
                                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                         ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
                                                             template<> struct ConwayPolynomial<2, 17> { using ZPZ = aerobus::zpz<2>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1»; }; // NOLINT
                                       template<> struct ConwayPolynomial<2, 18> { using ZPZ = aerobus::zpz<<>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
03172
                                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>; };
                                       template<> struct ConwayPolynomial<2, 19> { using ZPZ = aerobus::zpz<2>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , Z
                                         \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{1} >, \ \texttt{ZPZV} < \texttt{2} >, \ \texttt{2
                                        NOLINT
                                                                 template<> struct ConwayPolynomial<2, 20> { using ZPZ = aerobus::zpz<2>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1</pre>; };
                                        // NOLINT
 03175
                                                                  template<> struct ConwayPolynomial<3, 1> { using ZPZ = aerobus::zpz<3>; using type =
                                     POLYV<ZPZV<1>, ZPZV<1»; }; // NOLINT
                                                               template<> struct ConwayPolynomial<3, 2> { using ZPZ = aerobus::zpz<3>; using type =
                                     POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2»; }; // NOLINT
                                                                  template<> struct ConwayPolynomial<3, 3> { using ZPZ = aerobus::zpz<3>; using type =
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
                                                                  template<> struct ConwayPolynomial<3, 4> { using ZPZ = aerobus::zpz<3>; using type =
 03178
                                     POLYV<ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2»; }; // NOLINT
                                                               template<> struct ConwayPolynomial<3, 5> { using ZPZ = aerobus::zpz<3>; using type =
03179
                                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT
 03180
                                                                   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
 03181
                                                              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
                                    template<> struct ConwayPolynomial<3, 8> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT
 03182
                                                                 template<> struct ConwayPolynomial<3, 9> { using ZPZ = aerobus::zpz<3>; using type
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; //
                                                               template<> struct ConwayPolynomial<3, 10> { using ZPZ = aerobus::zpz<3>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<1>,
                                        ZPZV<2»: }: // NOLINT</pre>
                                                                  template<> struct ConwayPolynomial<3, 11> { using ZPZ = aerobus::zpz<3>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<0>, ZPZV<1»; }; // NOLINT</pre>
03186
                                                                  template<> struct ConwayPolynomial<3, 12> { using ZPZ = aerobus::zpz<3>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<0>,
                                        ZPZV<1>, ZPZV<0>, ZPZV<2»; }; // NOLINT</pre>
                                                                    template<> struct ConwayPolynomial<3, 13> { using ZPZ = aerobus::zpz<3>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
03188
                                                                   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<1>, ZPZV<2>, ZPZV<1>, ZPZV<1
                                                                  template<> struct ConwayPolynomial<3, 15> { using ZPZ = aerobus::zpz<3>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<1»; }; // NOLINT</pre>
03190
                                                             template<> struct ConwayPolynomial<3, 16> { using ZPZ = aerobus::zpz<3>; using type =
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<3, 17> { using ZPZ = aerobus::zpz<3>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<1»; }; // NOLINT</pre>
                                                             template<> struct ConwayPolynomial<3, 18> { using ZPZ = aerobus::zpz<3>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1 , ZPZV<1
                                       ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<1>, ZPZV<2>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>; }; // NOLINT
template<> struct ConwayPolynomial<3, 19> { using ZPZ = aerobus::zpz<3>; using type =
POLYV<ZPZV<1>, ZPZV<0>, Z
03193
                                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1»; };</pre>
                                       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<2>, ZPZV<2>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; };
                                         // NOLINT
                                                                  template<> struct ConwayPolynomial<5, 1> { using ZPZ = aerobus::zpz<5>; using type =
                                    POLYV<ZPZV<1>, ZPZV<3»; }; // NOLINT
 03196
                                                             template<> struct ConwayPolynomial<5, 2> { using ZPZ = aerobus::zpz<5>; using type =
                                    POLYV<ZPZV<1>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                                                 template<> struct ConwayPolynomial<5, 3> { using ZPZ = aerobus::zpz<5>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<3»; };
                                                                                                                                                                                                                               // NOLINT
                        template<> struct ConwayPolynomial<5, 4> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                          template<> struct ConwayPolynomial<5, 5> { using ZPZ = aerobus::zpz<5>; using type =
 03199
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<3»; }; // NOLINT
03200
                                           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
 03201
                                            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
 03202
                                         template<> struct ConwayPolynomial<5, 8> { using ZPZ = aerobus::zpz<5>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<5, 9> { using ZPZ = aerobus::zpz<5>; using type =
 03203
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<1>, ZPZV<3»; }; //
 03204
                                         template<> struct ConwayPolynomial<5, 10> { using ZPZ = aerobus::zpz<5>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2»; }; // NOLINT
                          template<> struct ConwayPolynomial<5, 11> { using ZPZ = aerobus::zpz<5>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
03205
                           ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                          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<1>, ZPZV<0>, ZPZV<4>, ZPZV<4>,
                           ZPZV<3>, ZPZV<2>, ZPZV<2»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<5, 13> { using ZPZ = aerobus::zpz<5>; using type =
03207
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<4>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<5, 14> { using ZPZ = aerobus::zpz<5>; using type =
03208
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4 , ZPZV<4
                          ZPZV<2>, ZPZV<3>, ZPZV<0>, ZPZV<1>, ZPZV<2»; }; // NOLINT
    template<> struct ConwayPolynomial<5, 15> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03209
                           ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<3»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<5, 16> { using ZPZ = aerobus::zpz<5>; using type =
03210
                           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>,
                          ZPZV<4>, ZPZV<4>, ZPZV<2>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2»; }; // NOLINT
  template<> struct ConwayPolynomial<5, 17> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03211
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<2>, ZPZV<3»; }; // NOLINT
                                            template<> struct ConwayPolynomial<5, 18> { using ZPZ = aerobus::zpz<5>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
                           template<> struct ConwayPolynomial<5, 19> { using ZPZ = aerobus::zpz<5>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<
03213
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<3»; }; //</pre>
                                          template<> struct ConwayPolynomial<5, 20> { using ZPZ = aerobus::zpz<5>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3
                           ZPZV<4>, ZPZV<3>, ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<0>, ZPZV<4>, ZPZV<1>, ZPZV<1>, ZPZV<2</pre>; };
                           // NOLINT
                                            template<> struct ConwayPolynomial<7, 1> { using ZPZ = aerobus::zpz<7>; using type =
03215
                         POLYV<ZPZV<1>, ZPZV<4»; }; // NOLINT
                                            template<> struct ConwayPolynomial<7, 2> { using ZPZ = aerobus::zpz<7>; using type =
                          POLYV<ZPZV<1>, ZPZV<6>, ZPZV<3»; }; // NOLINT
                                         template<> struct ConwayPolynomial<7, 3> { using ZPZ = aerobus::zpz<7>; using type =
 03217
                        POLYV<ZPZV<1>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT template<> struct ConwayPolynomial<7, 4> { using ZPZ = aerobus::zpz<7>; using type =
 03218
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<4>, ZPZV<3»; };
                                                                                                                                                                                                                                                                      // NOLINT
                                         template<> struct ConwayPolynomial<7, 5> { using ZPZ = aerobus::zpz<7>; using type =
 03219
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
 03220
                                            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<5>, ZPZV<5>, ZPZV<6>, ZPZV<5>; }; // NOLINT 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<6>, ZPZV<6>, ZPZV<4»; }; // NOLINT
03221
                                            template<> struct ConwayPolynomial<7, 8> { using ZPZ = aerobus::zpz<7>; using type =
                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<6>, ZPZV<2>, ZPZV<3»; };
 03223
                                         template<> struct ConwayPolynomial<7, 9> { using ZPZ = aerobus::zpz<7>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                          NOLINT
                          template<> struct ConwayPolynomial<7, 10> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<1>, ZPZV<2>, ZPZV<3>,
                           ZPZV<3»; }; // NOLINT</pre>
03225
                                         template<> struct ConwayPolynomial<7, 11> { using ZPZ = aerobus::zpz<7>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<4»; }; // NOLINT
                                            template<> struct ConwayPolynomial<7, 12> { using ZPZ = aerobus::zpz<7>; using type
03226
                           POLYV<ZPZV<1>, ZPZV<0>, 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</pre>
03227
                                         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<0>, ZPZV<6>, ZPZV<0>, ZPZV<4»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<7, 14> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , Z
03228
                           ZPZV<2>, ZPZV<0>, ZPZV<3>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<7, 15> { using ZPZ = aerobus::zpz<7>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<6>, ZPZV<6 , ZPZV<6
 03230
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<5>,
                                ZPZV<3>, ZPZV<4>, ZPZV<1>, ZPZV<6>, ZPZV<2>, ZPZV<4>, ZPZV<3»; }; // NOLINT
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<4>; }; // NOLINT
    template<> struct ConwayPolynomial<7, 18> { using ZPZ = aerobus::zpz<7>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<6>, ZPZV<6</pre>
                                 ZPZV<6>, ZPZV<5>, ZPZV<1>, ZPZV<3>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<2>, ZPZV<2>, ZPZV<3»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<7, 19> { using ZPZ = aerobus::zpz<7>; using type = POLYV<ZPZV<1>, ZPZV<0>, Z
                                NOLINT
                                                    template<> struct ConwayPolynomial<7, 20> { using ZPZ = aerobus::zpz<7>; using type
03234
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, 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<0>, ZPZV<1>, ZPZV<3>; };
                                 // NOLINT
03235
                                                   template<> struct ConwayPolynomial<11, 1> { using ZPZ = aerobus::zpz<11>; using type =
                               POLYV<ZPZV<1>, ZPZV<9»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<11, 2> { using ZPZ = aerobus::zpz<11>; using type =
                               POLYV<ZPZV<1>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<11, 3> { using ZPZ = aerobus::zpz<11>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<9»; }; // NOLINT template<> struct ConwayPolynomial<11, 4> { using ZPZ = aerobus::zpz<11>; using type =
03238
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<11, 5> { using ZPZ = aerobus::zpz<11>; using type =
03239
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<0>, ZPZV<9»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<11, 6> { using ZPZ = aerobus::zpz<11>; using type =
 03240
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<4>, ZPZV<6>, ZPZV<7>, ZPZV<2»; }; // NOLINT
                              template<> struct ConwayPolynomial<11, 7> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<9»; }; // NOLINT
template<> struct ConwayPolynomial<11, 8> { using ZPZ = aerobus::zpz<11>; using type =
 03241
03242
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<2>; }; // NOLINT
                                                    template<> struct ConwayPolynomial<11, 9> { using ZPZ = aerobus::zpz<11>; using type
 03243
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9
                                NOLINT
03244
                                                    template<> struct ConwayPolynomial<11, 10> { using ZPZ = aerobus::zpz<11>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<8>, ZPZV<10>, ZPZV<6>, ZPZV<6>,
                                ZPZV<2»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<11, 11> { using ZPZ = aerobus::zpz<11>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<10>, ZPZV<9»; }; // NOLINT
template<> struct ConwayPolynomial<11, 12> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<4>, ZPZV<4>, ZPZV<5>, ZPZV<5>,
03246
                                ZPZV<6>, ZPZV<5>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<11, 13> { using ZPZ = aerobus::zpz<11>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<9»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial<11, 14> { using ZPZ = aerobus::zpz<11>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                                     template<> struct ConwayPolynomial<11, 15> { using ZPZ = aerobus::zpz<11>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<7>, ZPZV<0>, ZPZV<5>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>; }; // NOLINT
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<1>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<10>, ZPZV<2>; }; // NOLINT
template<> struct ConwayPolynomial<11, 17> { using ZPZ = aerobus::zpz<11>; using type =
03250
                                POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                                03252
                                                     template<> struct ConwayPolynomial<11, 18> { using ZPZ = aerobus::zpz<11>; using type =
                               POLYYCZPZVC1>, ZPZVCO>, ZPZVCO
                                                    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<2>, ZPZV<2>, ZPZV<2</pre>; };
                                NOLINT
                               template<> struct ConwayPolynomial<11, 20> { using ZPZ = aerobus::zpz<11>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<5>, ZPZV<5</pre>
                                // NOLINT
                                                     template<> struct ConwayPolynomial<13, 1> { using ZPZ = aerobus::zpz<13>; using type =
                               POLYV<ZPZV<1>, ZPZV<11»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<13, 2> { using ZPZ = aerobus::zpz<13>; using type =
03256
                               POLYV<ZPZV<1>, ZPZV<12>, ZPZV<2»; }; // NOLINT
                                                    template<> struct ConwayPolynomial<13, 3> { using ZPZ = aerobus::zpz<13>; using type =
03257
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<13, 4> { using ZPZ = aerobus::zpz<13>; using type =
 03258
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<12>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 5> { using ZPZ = aerobus::zpz<13>; using type =
03259
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>; j; // NOLINT template<> struct ConwayPolynomial<13, 6> { using ZPZ = aerobus::zpz<13>; using type =
 03260
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<11>, ZPZV<11>, ZPZV<2»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<13, 7> { using ZPZ = aerobus::zpz<13>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<11»; };
 03262
                                                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 template<> struct ConwayPolynomial<13, 9> { using ZPZ = aerobus::zpz<13>; using type =
 03263
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<8>, ZPZV<12>, ZPZV<12>, ZPZV<11>; };
03264
                                                                 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</pre>
                                                                    template<> struct ConwayPolynomial<13, 11> { using ZPZ = aerobus::zpz<13>; using type =
03265
                                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           ZPZV<3>, ZPZV<11»; }; // NOLINT</pre>
                                                                 template<> struct ConwayPolynomial<13, 12> { using ZPZ = aerobus::zpz<13>; using type =
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<8>, ZPZV<11>, ZPZV<3>, ZPZV<1>, ZPZV<4>, ZPZV<4 , ZPZV<
                                                                  template<> struct ConwayPolynomial<13, 13> { using ZPZ = aerobus::zpz<13>; using type =
03267
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                               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<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<6>, ZPZV<6>, ZPZV<11>, ZPZV<7>, ZPZV<10>, ZPZV<10>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 15> { using ZPZ = aerobus::zpz<13>; using type =
03269
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<12>,
                                         ZPZV<2>, ZPZV<11>, ZPZV<10>, ZPZV<11>, ZPZV<8>, ZPZV<11»; }; // NOLINT
                                        template<> struct ConwayPolynomial<13, 16> { using ZPZ = aerobus::zpz<13>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                        ZPZV<8>, ZPZV<2>, ZPZV<12>, ZPZV<9>, ZPZV<12>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; // NOLINT
    template<> struct ConwayPolynomial<13, 17> { using ZPZ = aerobus::zpz<13>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0</pre>
03271
                                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<6>, ZPZV<61, ZPZV<11»; }; // NOLINT</pre>
                                                               template<> struct ConwayPolynomial<13, 18> { using ZPZ = aerobus::zpz<13>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<4>, ZPZV<11>, ZPZV<11>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<6>, ZPZV<6>, ZPZV<9>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<13, 19> { using ZPZ = aerobus::zpz<13>; using type =
03273
                                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           ZPZV<0>, ZPZV<0>
                                                                  template<> struct ConwayPolynomial<13, 20> { using ZPZ = aerobus::zpz<13>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5>, ZPZV<5 , ZPZV<5
                                           // NOLINT
                                                                    template<> struct ConwayPolynomial<17, 1> { using ZPZ = aerobus::zpz<17>; using type =
                                        POLYV<ZPZV<1>, ZPZV<14»; }; // NOLINT
                                                                 template<> struct ConwayPolynomial<17, 2> { using ZPZ = aerobus::zpz<17>; using type =
                                      POLYV<ZPZV<1>, ZPZV<16>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<17, 3> { using ZPZ = aerobus::zpz<17>; using type =
03277
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT template<> struct ConwayPolynomial<17, 4> { using ZPZ = aerobus::zpz<17>; using type =
 03278
                                      POLYV<2PZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<10>, ZPZV<10, ZPZV<3; }; // NOLINT template<> struct ConwayPolynomial<17, 5> { using ZPZ = aerobus::zpz<17>; using type =
 03279
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<14»; }; // NOLINT
                                      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
03280
                                                                   template<> struct ConwayPolynomial<17, 7> { using ZPZ = aerobus::zpz<17>; using type =
 03281
                                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<14»; }; // NOLINT
                                                                    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»; };
03283
                                                                  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<0>, ZPZV<7>, ZPZV<8>, ZPZV<14»; };
                                         // NOLINT
                                                                    template<> struct ConwayPolynomial<17, 10> { using ZPZ = aerobus::zpz<17>; using type
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>,
                                         ZPZV<3»; }; // NOLINT</pre>
03285
                                                                      template<> struct ConwayPolynomial<17, 11> { using ZPZ = aerobus::zpz<17>; using type
                                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                    template<> struct ConwayPolynomial<17, 12> { using ZPZ = aerobus::zpz<17>; using type
                                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<14>, ZPZV<14>, ZPZV<14>, ZPZV<13>, ZPZV<6>, ZPZV<6>, ZPZV<14>, ZPZV<14
03287
                                                               template<> struct ConwayPolynomial<17, 13> { using ZPZ = aerobus::zpz<17>; using type =
                                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1
, ZPZV
,
                                           ZPZV<16>, ZPZV<13>, ZPZV<9>, ZPZV<3>, ZPZV<3»; }; // NOLINT</pre>
03289
                                                                 template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::zpz<17>; using type =
                                        Template<> struct ConwayPolynomial<17, 15> { using ZPZ = aerobus::ZPZV17>; 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<6>, ZPZV<14>, ZPZV<14>, ZPZV<14>; // NOLINT template<> struct ConwayPolynomial<17, 16> { using ZPZ = aerobus::ZpZ<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1 ,
03290
                                         ZPZV<5>, ZPZV<2>, ZPZV<12>, ZPZV<13>, ZPZV<12>, ZPZV<1>, ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<17, 17> { using ZPZ = aerobus::zpz<17>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03291
                                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<14»; }; // NOLINT
template<> struct ConwayPolynomial<17, 18> { using ZPZ = aerobus::zpz<17>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1
03292
                                         ZPZV<7>, ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<11>, ZPZV<13>, ZPZV<13>, ZPZV<9>, ZPZV<3»; }; // NOLINT</pre>
                                                                 template<> struct ConwayPolynomial<17, 19> { using ZPZ = aerobus::zpz<17>; using type =
                                         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<114»; }; //</pre>
                                         NOLTNT
```

```
template<> struct ConwayPolynomial<17, 20> { using ZPZ = aerobus::zpz<17>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>,
                        ZPZV<16>, ZPZV<14>, ZPZV<13>, ZPZV<3>, ZPZV<14>, ZPZV<9>, ZPZV<1>, ZPZV<13>, ZPZV<2>, ZPZV<5>,
                       ZPZV<3»; }; // NOLINT</pre>
03295
                                       template<> struct ConwayPolynomial<19, 1> { using ZPZ = aerobus::zpz<19>; using type =
                      POLYV<ZPZV<1>, ZPZV<17»; }; // NOLINT
                                       template<> struct ConwayPolynomial<19, 2> { using ZPZ = aerobus::zpz<19>; using type =
                       POLYV<ZPZV<1>, ZPZV<18>, ZPZV<2»; }; // NOLINT
                                     template<> struct ConwayPolynomial<19, 3> { using ZPZ = aerobus::zpz<19>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<19, 4> { using ZPZ = aerobus::zpz<19>; using type =
03298
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<11>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 5> { using ZPZ = aerobus::zpz<19>; using type =
03299
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<17»; }; // NOLINT
03300
                                     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<6>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 7> { using ZPZ = aerobus::zpz<19>; using type =
03301
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<3>, ZPZV<3>, ZPZV<3>; };
                       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<16>, ZPZV<17»; };
                        // NOLINT
                       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>,
03304
                       ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<19, 11> { using ZPZ = aerobus::zpz<19>; using type =
03305
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<8>, ZPZV<17»; }; // NOLINT
    template<> struct ConwayPolynomial<19, 12> { using ZPZ = aerobus::zpz<19>; using type =
03306
                       POLYY<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
                                       template<> struct ConwayPolynomial<19, 13> { using ZPZ = aerobus::zpz<19>; using type =
03307
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<17»; }; // NOLINT
  template<> struct ConwayPolynomial<19, 14> { using ZPZ = aerobus::zpz<19>; using type
03308
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<0>, ZPZV<1>, ZPZV<1
                                        template<> struct ConwayPolynomial<19, 15> { using ZPZ = aerobus::zpz<19>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZ
                        \mbox{ZPZV}<11>, \mbox{ZPZV}<13>, \mbox{ZPZV}<15>, \mbox{ZPZV}<14>, \mbox{ZPZV}<0>, \mbox{ZPZV}<17»; \mbox{} ; \mbox{} // \mbox{NOLINT} 
                       template<> struct ConwayPolynomial<19, 16> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03310
                       ZPZV<13>, ZPZV<0>, ZPZV<15>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<14>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<19, 17> { using ZPZ = aerobus::zpz<19>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       template<> struct ConwayPolynomial<19, 18> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<2>; }; // NOLINT
                                       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<18>, ZPZV<17»; }; //</pre>
                       template<> struct ConwayPolynomial<19, 20> { using ZPZ = aerobus::zpz<19>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<6>, ZPZV<13>, ZPZV<6>, ZPZV<11>, ZPZV<11>, ZPZV<2»;
03314
                                       template<> struct ConwayPolynomial<23, 1> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<18»; }; // NOLINT
                                      template<> struct ConwayPolynomial<23, 2> { using ZPZ = aerobus::zpz<23>; using type =
03316
                      POLYV<ZPZV<1>, ZPZV<21>, ZPZV<5»; }; // NOLINT
03317
                                        template<> struct ConwayPolynomial<23, 3> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18»; }; // NOLINT template<> struct ConwayPolynomial<23, 4> { using ZPZ = aerobus::zpz<23>; using type =
03318
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<19>, ZPZV<5»; }; // NOLINT
                      template<> struct ConwayPolynomial<23, 5> { using ZPZ = aerobus::zpz<23>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<18»; }; // NOLINT
template<> struct ConwayPolynomial<23, 6> { using ZPZ = aerobus::zpz<23>; using type =
03319
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<9>, ZPZV<9>, ZPZV<1>, ZPZV<5»; }; // NOLINT
                                       template<> struct ConwayPolynomial<23, 7> { using ZPZ = aerobus::zpz<23>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<18»; }; // NOLINT
03322
                                       template<> struct ConwayPolynomial<23, 8> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<20>, ZPZV<5>, ZPZV<5>, ZPZV<5>; };
                                       template<> struct ConwayPolynomial<23, 9> { using ZPZ = aerobus::zpz<23>; using type
03323
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<8>, ZPZV<8>, ZPZV<9>, ZPZV<18»; };
                       // NOLINT
03324
                                      template<> struct ConwayPolynomial<23, 10> { using ZPZ = aerobus::zpz<23>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<15>, ZPZV<6>, ZPZV<1>,
                       ZPZV<5»: 1: // NOLINT
                                        template<> struct ConwayPolynomial<23, 11> { using ZPZ = aerobus::zpz<23>; using type
03325
                       POLYYCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<7>, ZPZV<18»; }; // NOLINT
                                     template<> struct ConwayPolynomial<23, 12> { using ZPZ = aerobus::zpz<23>; using type
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<21>, ZPZV<21>, ZPZV<15>, ZPZV<14>, ZPZV<12>, ZPZV<18>, ZPZV<12>, ZPZV<18>, ZPZV<18>, ZPZV<18>, ZPZV<18>, ZPZV<18>, ZPZV<18>, ZPZV<18>, ZPZV<18>, ZPZV<18
03327
                                     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<0>, ZPZV<9>, ZPZV<18»; };</pre>
                                                                                                                                                                                                                                      // NOLINT
                                                 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<1>, ZPZV<5>, ZPZV<1>,
                             ZPZV<18>, ZPZV<19>, ZPZV<1>, ZPZV<22>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<23, 15> { using ZPZ = aerobus::zpz<23>; using type =
03329
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<8>, ZPZV<15>, ZPZV<9>, ZPZV<7>, ZPZV<18>, ZPZV<18»; }; // NOLINT
template<> struct ConwayPolynomial<23, 16> { using ZPZ = aerobus::zpz<23>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1 , ZPZV<1
03331
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<18»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<23, 18> { using ZPZ = aerobus::zpz<23>; using type
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<18>, ZPZV<2>, ZPZV<1>, ZPZV<11>, ZPZV<18>, ZPZV<2>, ZPZV<5>; }; // NOLINT template<> struct ConwayPolynomial<23, 19> { using ZPZ = aerobus::zpz<23>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZP
03333
                               ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<18»; }; //</pre>
03334
                                                 template<> struct ConwayPolynomial<29, 1> { using ZPZ = aerobus::zpz<29>; using type =
                             POLYV<ZPZV<1>, ZPZV<27»; }; // NOLINT
                                                template<> struct ConwayPolynomial<29, 2> { using ZPZ = aerobus::zpz<29>; using type =
03335
                             POLYV<ZPZV<1>, ZPZV<24>, ZPZV<2»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<29, 3> { using ZPZ = aerobus::zpz<29>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<27»; }; // NOLINT
                                                template<> struct ConwayPolynomial<29, 4> { using ZPZ = aerobus::zpz<29>; using type =
03337
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<15>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<29, 5> { using ZPZ = aerobus::zpz<29>; using type =
03338
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<27»; }; // NOLINT
03339
                                                  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
                                              template<> struct ConwayPolynomial<29, 7> { using ZPZ = aerobus::zpz<29>; using type =
03340
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<27»; }; // NOLINT template<> struct ConwayPolynomial<29, 8> { using ZPZ = aerobus::zpz<29>; using type =
03341
                              POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<24>, ZPZV<26>, ZPZV<23>, ZPZV<2»; }; //
                                                 template<> struct ConwayPolynomial<29, 9> { using ZPZ = aerobus::zpz<29>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<22>, ZPZV<22>, ZPZV<27»; };
                              // NOLINT
                             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<8>, ZPZV<17>, ZPZV<2>, ZPZV<22>,
03343
                              ZPZV<2»; }; // NOLINT</pre>
                                                 template<> struct ConwayPolynomial<29, 11> { using ZPZ = aerobus::zpz<29>; using type
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<8>, ZPZV<27»; }; // NOLINT</pre>
                             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<28>, ZPZV<16>, ZPZV<25>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>; }; // NOLINT
                                                 template<> struct ConwayPolynomial<29, 13> { using ZPZ = aerobus::zpz<29>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<27»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<29, 14> { using ZPZ = aerobus::zpz<29>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<3>, ZPZV<14>, ZPZV<10>,
03347
                             ZPZV<21>, ZPZV<18>, ZPZV<27>, ZPZV<5>, ZPZV<2»; }; // NOLINT
   template<> struct ConwayPolynomial<29, 15> { using ZPZ = aerobus::zpz<29>; using type
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<14>, ZPZV<8>, ZPZV<1>, ZPZV<12, ZPZV<26>, ZPZV<27»; }; // NOLINT
template<> struct ConwayPolynomial<29, 16> { using ZPZ = aerobus::zpz<29>; using type
03349
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<2 >, // NOLINT
                                              template<> struct ConwayPolynomial<29, 18> { using ZPZ = aerobus::zpz<29>; using type =
03351
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<1>, ZPZV<1>
                             ZPZV<6>, ZPZV<26>, ZPZV<2>, ZPZV<10>, ZPZV<45>, ZPZV<16>, ZPZV<6>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<29, 19> { using ZPZ = aerobus::zpz<29>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<0>, ZPZV<0>
03353
                                                 template<> struct ConwayPolynomial<31, 1> { using ZPZ = aerobus::zpz<31>; using type =
                             POLYV<ZPZV<1>, ZPZV<28»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<31, 2> { using ZPZ = aerobus::zpz<31>; using type =
03354
                             POLYV<ZPZV<1>, ZPZV<29>, ZPZV<3»; }; // NOLINT
                                                template<> struct ConwayPolynomial<31, 3> { using ZPZ = aerobus::zpz<31>; using type =
03355
                             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT template<> struct ConwayPolynomial<31, 4> { using ZPZ = aerobus::zpz<31>; using type =
03356
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<16>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<31, 5> { using ZPZ = aerobus::zpz<31>; using type =
03357
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<28»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<31, 6> { using ZPZ = aerobus::zpz<31>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<16>, ZPZV<8>, ZPZV<3»; }; // NOLINT
                          template<> struct ConwayPolynomial<31, 7> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28»; }; // NOLINT
template<> struct ConwayPolynomial<31, 8> { using ZPZ = aerobus::zpz<31>; using type =
03359
03360
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<12>, ZPZV<24>, ZPZV<3»; }; //
                                    template<> struct ConwayPolynomial<31, 9> { using ZPZ = aerobus::zpz<31>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<20, ZPZV<29>, ZPZV<28»; };
                        // NOLINT
                                       template<> struct ConwayPolynomial<31, 10> { using ZPZ = aerobus::zpz<31>; using type =
03362
                        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3 , ZPZV<3
                        ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<31, 11> { using ZPZ = aerobus::zpz<31>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<20>, ZPZV<28»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<31, 12> { using ZPZ = aerobus::zpz<31>; using type
03364
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<14>, ZPZV<28>, ZPZV<2>, ZPZV<9>, ZPZV<25>, ZPZV<12>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial<31, 13> { using ZPZ = aerobus::zpz<31>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<28»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<31, 14> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1 , ZPZV<1 ,
03366
                        ZPZV<1>, ZPZV<18>, ZPZV<18>, ZPZV<6>, ZPZV<3»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<31, 15> { using ZPZ = aerobus::zpz<31>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<29>, ZPZV<12>, ZPZV<13>, ZPZV<23>, ZPZV<25>, ZPZV<28*; }; // NOLINT
   template<> struct ConwayPolynomial<31, 16> { using ZPZ = aerobus::zpz<31>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<24>, ZPZV<25, ZPZV<28>, ZPZV<11>, ZPZV<19>, ZPZV<27>, ZPZV<3*; }; // NOLINT
   template<> struct ConwayPolynomial<31, 17> { using ZPZ = aerobus::zpz<31>; using type =
03368
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                        template<> struct ConwayPolynomial<31, 18> { using ZPZ = aerobus::zpz<31>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<27>, ZPZV<27>, ZPZV<24>, ZPZV<24>, ZPZV<25>, ZPZV<25>, ZPZV<3>, ZPZV<3»; }; // NOLINT
03370
                                      template<> struct ConwayPolynomial<31, 19> { using ZPZ = aerobus::zpz<31>; using type
03371
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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»; }; //</pre>
                        NOLINT
03372
                                      template<> struct ConwayPolynomial<37, 1> { using ZPZ = aerobus::zpz<37>; using type =
                       POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                       template<> struct ConwayPolynomial<37, 2> { using ZPZ = aerobus::zpz<37>; using type =
                       POLYV<ZPZV<1>, ZPZV<33>, ZPZV<2»; }; // NOLINT
03374
                                     template<> struct ConwayPolynomial<37, 3> { using ZPZ = aerobus::zpz<37>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<37, 4> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<24>, ZPZV<2»; }; // NOLINT
03375
                                        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
03377
                                    template<> struct ConwayPolynomial<37, 6> { using ZPZ = aerobus::zpz<37>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<4>, ZPZV<30>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<37, 7> { using ZPZ = aerobus::zpz<37>; using type =
03378
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<35»; }; // NOLINT
03379
                                      template<> struct ConwayPolynomial<37, 8> { using ZPZ = aerobus::zpz<37>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<20>, ZPZV<27>, ZPZV<27>, ZPZV<27>, ZPZV<28; };
03380
                                     template<> struct ConwayPolynomial<37, 9> { using ZPZ = aerobus::zpz<37>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<20>, ZPZV<32>, ZPZV<35»; };
                        // NOLINT
03381
                                       template<> struct ConwayPolynomial<37, 10> { using ZPZ = aerobus::zpz<37>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<29>, ZPZV<18>, ZPZV<18>, ZPZV<11>, ZPZV<4>,
                        ZPZV<2»: }: // NOLINT
                       template<> struct ConwayPolynomial<37, 11> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                        ZPZV<2>, ZPZV<35»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<37, 12> { using ZPZ = aerobus::zpz<37>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<31>, ZPZV<10>, ZPZV<23>, ZPZV<23>, ZPZV<23>, ZPZV<23>, ZPZV<33>, ZPZV<33>, ZPZV<23>, ZPZV<33>, ZPZV<33
                                    template<> struct ConwayPolynomial<37, 13> { using ZPZ = aerobus::zpz<37>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; };</pre>
                                     V<0>, ZPZV<0>, ZPZV<6>, ZPZV<35»; }; // NOLINT
template<> struct ConwayPolynomial<37, 14> { using ZPZ = aerobus::zpz<37>; using type =
03385
                        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</pre>
                                       template<> struct ConwayPolynomial<37, 15> { using ZPZ = aerobus::zpz<37>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<31>,
                       ZPZV<28>, ZPZV<27>, ZPZV<13>, ZPZV<34>, ZPZV<33>, ZPZV<35»; }; // NOLINT
    template<> struct ConwayPolynomial<37, 17> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03387
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<35»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<37, 18> { using ZPZ = aerobus::zpz<37>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<8>, ZPZV<8>, ZPZV<15>,
                       ZPZV<1>, ZPZV<22>, ZPZV<20>, ZPZV<12>, ZPZV<32>, ZPZV<14>, ZPZV<27>, ZPZV<20>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<37, 19> { using ZPZ = aerobus::zpz<37>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
03389
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<23>, ZPZV<35»; }; //</pre>
03390
                                      template<> struct ConwayPolynomial<41, 1> { using ZPZ = aerobus::zpz<41>; using type =
                     POLYV<ZPZV<1>, ZPZV<35»; }; // NOLINT
                                       template<> struct ConwayPolynomial<41, 2> { using ZPZ = aerobus::zpz<41>; using type =
03391
                        POLYV<ZPZV<1>, ZPZV<38>, ZPZV<6»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<41, 3> { using ZPZ = aerobus::zpz<41>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<35»; }; // NOLINT template<> struct ConwayPolynomial<41, 4> { using ZPZ = aerobus::zpz<41>; using type =
03393
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<6»; }; // NOLINT
                     template<> struct ConwayPolynomial<41, 5> { using ZPZ = aerobus::zpz<41>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<40>, ZPZV<14>, ZPZV<35»; }; // NOLINT
03394
                                     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»; };
03396
                                   template<> struct ConwayPolynomial<41, 7> { using ZPZ = aerobus::zpz<41>; using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>; };
03397
                                   template<> struct ConwayPolynomial<41, 8> { using ZPZ = aerobus::zpz<41>; using type =
                     POLYV<ZPZV<1>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<5>, ZPZV<5>, ZPZV<52>, ZPZV<32>, ZPZV<62>, ZPZV<6>; ; template<> struct ConwayPolynomial<41, 9> { using ZPZ = aerobus::zpz<41>; using type =
03398
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<31>, ZPZV<5>, ZPZV<35»; };
                       // NOLINT
                                    template<> struct ConwayPolynomial<41, 10> { using ZPZ = aerobus::zpz<41>; using type =
03399
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<31>, ZPZV<8>, ZPZV<80, ZPZV<30>,
                      ZPZV<6»; }; // NOLINT
                                     template<> struct ConwayPolynomial<41, 11> { using ZPZ = aerobus::zpz<41>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                      ZPZV<20>, ZPZV<35»; }; // NOLINT</pre>
03401
                                     template<> struct ConwayPolynomial<41, 12> { using ZPZ = aerobus::zpz<41>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<24>, ZPZV<26>, ZPZV<34>, ZPZV<24>, ZPZV<27>, ZPZV<27>, ZPZV<6>; }; // NOLINT
03402
                                     template<> struct ConwayPolynomial<41, 13> { using ZPZ = aerobus::zpz<41>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   template<> struct ConwayPolynomial<41, 14> { using ZPZ = aerobus::zpz<41>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<5, 
                     ZPZV<27>, ZPZV<11>, ZPZV<39>, ZPZV<10>, ZPZV<6*; }; // NOLINT
    template<> struct ConwayPolynomial<41, 15> { using ZPZ = aerobus::zpz<41>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
                       ZPZV<16>, ZPZV<2>, ZPZV<35>, ZPZV<10>, ZPZV<21>, ZPZV<35»; }; // NOLINT</pre>
                                  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<4>, ZPZV<35»; }; // NOLINT</pre>
                     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<2>, ZPZV<2>, ZPZV<5, ZPZV<2>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>; // NOLINT
03406
                     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<10>, ZPZV<10>, ZPZV<35»; }; //</pre>
                      NOLINT
                                    template<> struct ConwayPolynomial<43, 1> { using ZPZ = aerobus::zpz<43>; using type =
                     POLYV<ZPZV<1>, ZPZV<40»; }; // NOLINT
03409
                                     template<> struct ConwayPolynomial<43, 2> { using ZPZ = aerobus::zpz<43>; using type =
                     POLYV<ZPZV<1>, ZPZV<42>, ZPZV<3»; }; // NOLINT
03410
                                   template<> struct ConwayPolynomial<43, 3> { using ZPZ = aerobus::zpz<43>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<40»; }; // NOLINT
                                     template<> struct ConwayPolynomial<43, 4> { using ZPZ = aerobus::zpz<43>; using type =
03411
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<42>, ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<43, 5> { using ZPZ = aerobus::zpz<43>; using type =
03412
                     03413
                                    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 template<> struct ConwayPolynomial<43, 7> { using ZPZ = aerobus::zpz<43>; using type
03414
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<7>, ZPZV<7>, ZPZV<40»; }; // NOLINT
                                   template<> struct ConwayPolynomial<43, 8> { using ZPZ = aerobus::zpz<43>; using type =
03415
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<39>, ZPZV<20>, ZPZV<24>, ZPZV<3*, };
                      NOLINT
03416
                                   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<1>, ZPZV<39>, ZPZV<39>, ZPZV<40»; };
                      // NOLINT
                                     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<57>, ZPZV<27>, ZPZV<24>,
                      ZPZV<3»; }; // NOLINT</pre>
                     template<> struct ConwayPolynomial<43, 11> { using ZPZ = aerobus::zpz<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<10
03418
                                     template<> struct ConwayPolynomial<43, 12> { using ZPZ = aerobus::zpz<43>; using type =
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<27>, ZPZV<16>, ZPZV<17>, ZPZV<6>, ZPZV<6>, ZPZV<28, ZPZV<28, ZPZV<28, ZPZV<38, ZP
03420
                                   template<> struct ConwayPolynomial<43, 13> { using ZPZ = aerobus::zpz<43>; using type
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<43, 14> { using ZPZ = aerobus::zpz<43>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<24>,
                      ZPZV<37>, ZPZV<18>, ZPZV<4>, ZPZV<19>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<43, 15> { using ZPZ = aerobus::zpz<43>; using type =
                     POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<40>; ; // NOLINT template<> struct ConwayPolynomial<43, 17> { using ZPZ = aerobus::zpz<43>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<36>, ZPZV<40»; };</pre>
                                                                                                                                                                                                                                                                                                                   // NOLINT
                     template<> struct ConwayPolynomial<43, 18> { using ZPZ = aerobus::2pZ<43>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<38>, ZPZV<41>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<24>, ZPZV<38, ZPZV<38>; // NOLINT
```

```
template<> struct ConwayPolynomial<43, 19> { using ZPZ = aerobus::zpz<43>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                    ZPZV<0>, ZPZV<0>
                                    NOLINT
 03426
                                                          template<> struct ConwayPolynomial<47, 1> { using ZPZ = aerobus::zpz<47>; using type =
                                   POLYV<ZPZV<1>, ZPZV<42»; }; // NOLINT
                                                           template<> struct ConwayPolynomial<47, 2> { using ZPZ = aerobus::zpz<47>; using type =
                                   POLYV<ZPZV<1>, ZPZV<45>, ZPZV<5»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<47, 3> { using ZPZ = aerobus::zpz<47>; using type =
 03428
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<42»; }; // NOLINT template<> struct ConwayPolynomial<47, 4> { using ZPZ = aerobus::zpz<47>; using type =
03429
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<40>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<47, 5> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42»; }; // NOLINT
 03430
 03431
                                                        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 template<> struct ConwayPolynomial<47, 7> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03432
                                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<19>, ZPZV<3>, ZPZV<5»; };
                                   template<> struct ConwayPolynomial<47, 9> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<42»; };
                                     // NOLINT
                                   template<> struct ConwayPolynomial<47, 10> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<42>, ZPZV<14>, ZPZV<14>, ZPZV<18>, ZPZV<45>, ZPZV<45>,
03435
                                    ZPZV<5»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<47, 11> { using ZPZ = aerobus::zpz<47>; using type =
03436
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<6>, ZPZV<42»; }; // NOLINT
    template<> struct ConwayPolynomial<47, 12> { using ZPZ = aerobus::zpz<47>; using type =
03437
                                   POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46>, ZPZV<46>, ZPZV<40>, ZPZV<46>, ZPZV<46>, ZPZV<40>, ZPZV<5»; }; // NOLINT
                                                           template<> struct ConwayPolynomial<47, 13> { using ZPZ = aerobus::zpz<47>; using type =
03438
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<42»; };</pre>
                                                                                                                                                                                                                                                                                     // NOLINT
                                                            template<> struct ConwayPolynomial<47, 14> { using ZPZ = aerobus::zpz<47>; using type
03439
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>,
                                                            template<> struct ConwayPolynomial<47, 15> { using ZPZ = aerobus::zpz<47>; using type
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<31>, ZPZV<14>, ZPZV<42>, ZPZV<13>, ZPZV<17>, ZPZV<42»; }; // NOLINT
    template<> struct ConwayPolynomial<47, 17> { using ZPZ = aerobus::zpz<47>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 
0.3441
                                   ZPZV<0>, ZPZ
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<4+, ZPZV<4
                                     \texttt{ZPZV<26>, \ ZPZV<44>, \ ZPZV<22>, \ ZPZV<11>, \ ZPZV<5>, \ ZPZV<45>, \ ZPZV<33>, \ ZPZV<5>; \ // \ \texttt{NOLINT} } 
                                   template<> struct ConwayPolynomial<47, 19> { using ZPZ = aerobus::zpz<47>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                                    NOLINT
                                                            template<> struct ConwayPolynomial<53, 1> { using ZPZ = aerobus::zpz<53>; using type =
                                   POLYV<ZPZV<1>, ZPZV<51»; }; // NOLINT
                                                         template<> struct ConwayPolynomial<53, 2> { using ZPZ = aerobus::zpz<53>; using type =
 03445
                                  POLYV<ZPZV<1>, ZPZV<49>, ZPZV<2»; }; // NOLINT
                                                           template<> struct ConwayPolynomia1<53, 3> { using ZPZ = aerobus::zpz<53>; using type =
 03446
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<51»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<53, 4> { using ZPZ = aerobus::zpz<53>; using type =
 03447
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<38>, ZPZV<2»; }; // NOLINT
 03448
                                                            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 template<> struct ConwayPolynomial<53, 6> { using ZPZ = aerobus::zpz<53>; using type =
03449
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<7>, ZPZV<4>, ZPZV<45>, ZPZV<2»; }; // NOLINT
                                                            template<> struct ConwayPolynomial<53, 7> { using ZPZ = aerobus::zpz<53>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51»; };
 03451
                                                        template<> struct ConwayPolynomial<53, 8> { using ZPZ = aerobus::zpz<53>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<29>, ZPZV<18>, ZPZV<18>, ZPZV<1>, ZPZV<2»; };
                                                         template<> struct ConwayPolynomial<53, 9> { using ZPZ = aerobus::zpz<53>; using type =
03452
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<51»; };
                                    // NOLINT
                                                           template<> struct ConwayPolynomial<53, 10> { using ZPZ = aerobus::zpz<53>; using type =
03453
                                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<2>, ZPZV<25>, ZPZV<29>,
                                    ZPZV<2»; }; // NOLINT</pre>
                                                           template<> struct ConwayPolynomial<53, 11> { using ZPZ = aerobus::zpz<53>; using type
03454
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                        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<10>, ZPZV<42>, ZPZV<34>, ZPZV<41>, ZPZV<41>, ZPZV<2»; }; // NOLINT
03456
                                                        template<> struct ConwayPolynomial<53, 13> { using ZPZ = aerobus::zpz<53>; using type =
                                   POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<0>, ZPZV<52>, ZPZV<28>, ZPZV<51»; }; // NOLINT
   template<> struct ConwayPolynomial<53, 14> { using ZPZ = aerobus::zpz<53>; using type
                                   POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<45>, ZPZV<45>, ZPZV<23>, ZPZV<52>,
                                   ZPZV<0>, ZPZV<37>, ZPZV<12>, ZPZV<23>, ZPZV<2»; }; // NOLINT
   template<> struct ConwayPolynomial<53, 15> { using ZPZ = aerobus::zpz<53>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
 03458
```

```
ZPZV<31>, ZPZV<15>, ZPZV<11>, ZPZV<20>, ZPZV<4>, ZPZV<51»; }; // NOLINT
                           template<> struct ConwayPolynomial<53, 17> { using ZPZ = aerobus::zpz<53>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
                           03460
                           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<5>, ZPZV<5>, ZPZV<51>,
                           ZPZV<27>, ZPZV<0>, ZPZV<39>, ZPZV<44>, ZPZV<6>, ZPZV<8>, ZPZV<16>, ZPZV<11>, ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<53, 19> { using ZPZ = aerobus::zpz<53>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<51»; }; //</pre>
                           NOLINT
                                           template<> struct ConwayPolynomial<59, 1> { using ZPZ = aerobus::zpz<59>; using type =
03462
                           POLYV<ZPZV<1>, ZPZV<57»; }; // NOLINT
                                              template<> struct ConwayPolynomial<59, 2> { using ZPZ = aerobus::zpz<59>; using type =
                           POLYV<ZPZV<1>, ZPZV<58>, ZPZV<2»; }; // NOLINT
 03464
                                           template<> struct ConwayPolynomial<59, 3> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<57»; }; // NOLINT template<> struct ConwayPolynomial<59, 4> { using ZPZ = aerobus::zpz<59>; using type =
03465
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<40>, ZPZV<2»; }; // NOLINT
                                           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
 03467
                                           template<> struct ConwayPolynomial<59, 6> { using ZPZ = aerobus::zpz<59>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<18>, ZPZV<38>, ZPZV<0>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<59, 7> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<57»; }; // NOLINT
03468
                                           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
                           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<32>, ZPZV<47>, ZPZV<47>, ZPZV<57»; };</pre>
 03470
                           // NOLINT
                                              template<> struct ConwayPolynomial<59, 10> { using ZPZ = aerobus::zpz<59>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<25>, ZPZV<4>, ZPZV<39>, ZPZV<15>,
                           ZPZV<2»; }; // NOLINT</pre>
                           \label{template} $$ \text{template}$ <> $$ \text{struct ConwayPolynomial}$ <> 9, 11> { using ZPZ = aerobus::zpz<59>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03472
                           ZPZV<6>, ZPZV<57»; };</pre>
                                                                                                                                  // NOLINT
                                             template<> struct ConwayPolynomial<59, 12> { using ZPZ = aerobus::zpz<59>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<39>, ZPZV<25>, ZPZV<51>, ZPZV<21>, ZPZV<38>, ZPZV<8>, ZPZV<1>, ZPZV<2»; }; // NOLINT
03474
                                           template<> struct ConwayPolynomial<59, 13> { using ZPZ = aerobus::zpz<59>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>. ZPZV<0>. ZPZV<3>. ZPZV<57»: }:
                                                                                                                                                                                                                 // NOLINT
                                            template<> struct ConwayPolynomial<59, 14> { using ZPZ = aerobus::zpz<59>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<51>, ZPZV<11>,
                            ZPZV<13>, ZPZV<25>, ZPZV<32>, ZPZV<26>, ZPZV<2»; }; // NOLINT</pre>
03476
                                          template<> struct ConwayPolynomial<59, 15> { using ZPZ = aerobus::zpz<59>; using type =
                          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<57>, ZPZV<27, ZPZV<23>, ZPZV<33>, ZPZV<33>, ZPZV<33>, ZPZV<33>, ZPZV<33>, ZPZV<33>, ZPZV<33>, ZPZV<34>, ZPZV<35>, ZPZV<35, ZPZV
03477
                            \texttt{ZPZV} < \texttt{0>, } \texttt{ZPZV} < \texttt{57} \text{, } ; \text{ } // \text{ NoLint } \texttt{Notion } \texttt{Notio
03478
                                          template<> struct ConwayPolynomial<59, 18> { using ZPZ = aerobus::zpz<59>; using type
                           Template<> struct ConwayPolynomial

POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<37>, ZPZV<37>, ZPZV<38>, ZPZV<27>,
ZPZV<11>, ZPZV<14>, ZPZV<14>, ZPZV<44>, ZPZV<46>, ZPZV<47>, ZPZV<47>, ZPZV<34>, ZPZV<32>, ZPZV<29; }; // NOLINT template<> struct ConwayPolynomial</pr>

POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , 
03479
                            ZPZV<0>, ZPZV<0>
03480
                                            template<> struct ConwayPolynomial<61, 1> { using ZPZ = aerobus::zpz<61>; using type =
                          POLYV<ZPZV<1>, ZPZV<59»; }; // NOLINT
                                           template<> struct ConwayPolynomial<61, 2> { using ZPZ = aerobus::zpz<61>; using type =
03481
                           POLYV<ZPZV<1>, ZPZV<60>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<61, 3> { using ZPZ = aerobus::zpz<61>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<59»; };
                                                                                                                                                                                                                                            // NOLINT
                                           template<> struct ConwayPolynomial<61, 4> { using ZPZ = aerobus::zpz<61>; using type =
 03483
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<40>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<61, 5> { using ZPZ = aerobus::zpz<61>; using type =
03484
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<59»; }; // NOLINT
 03485
                                              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
 03486
                                          template<> struct ConwayPolynomial<61, 7> { using ZPZ = aerobus::zpz<61>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5), ZPZV<5, ZPZV<5, ZPZV<5
 03487
                                             template<> struct ConwayPolynomial<61, 8> { using ZPZ = aerobus::zpz<61>; using type =
                          POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<1>, ZPZV<56>, ZPZV<2»; };
                                             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<5>, ZPZV<5>, ZPZV<5>, ZPZV<50>, ZPZV<59»; };
                                           template<> struct ConwayPolynomial<61, 10> { using ZPZ = aerobus::zpz<61>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<15>, ZPZV<44>, ZPZV<16>, ZPZV<6>,
                           ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<61, 11> { using ZPZ = aerobus::zpz<61>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<18>, ZPZV<59»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<61, 12> { using ZPZ = aerobus::zpz<61>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<42>, ZPZV<33>, ZPZV<8>, ZPZV<38>, ZPZV<14>, ZPZV<15>, ZPZV<15>, ZPZV<2»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<61, 13> { using ZPZ = aerobus::zpz<61>; using type
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                 template<> struct ConwayPolynomial<61, 14> { using ZPZ = aerobus::zpz<61>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<48>, ZPZV<26>, ZPZV<11>,
                                ZPZV<8>, ZPZV<30>, ZPZV<54>, ZPZV<48>, ZPZV<2»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<61, 15> { using ZPZ = aerobus::zpz<61>; using type
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                               ZPZV<35>, ZPZV<44>, ZPZV<25>, ZPZV<23>, ZPZV<51>, ZPZV<59»; }; // NOLINT
template<> struct ConwayPolynomial<61, 17> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>
03495
                               template<> struct ConwayPolynomial<61, 18> { using ZPZ = aerobus::zpz<61>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<35>, ZPZV<36>, ZPZV<13>,
03496
                                 ZPZV<36>, ZPZV<4>, ZPZV<32>, ZPZV<57>, ZPZV<42>, ZPZV<25>, ZPZV<25>, ZPZV<52>, ZPZV<52
                               template<> struct ConwayPolynomial<61, 19> { using ZPZ = aerobus::zpz<61>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03497
                                 ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<59»; };</pre>
                                                    template<> struct ConwayPolynomial<67, 1> { using ZPZ = aerobus::zpz<67>; using type =
                               POLYV<ZPZV<1>, ZPZV<65»; };
                                                                                                                                                                                  // NOLINT
                                                    template<> struct ConwayPolynomial<67, 2> { using ZPZ = aerobus::zpz<67>; using type =
03499
                               POLYV<ZPZV<1>, ZPZV<63>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<67, 3> { using ZPZ = aerobus::zpz<67>; using type =
03500
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<65»; }; // NOLINT
                                                     template<> struct ConwayPolynomial<67, 4> { using ZPZ = aerobus::zpz<67>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<54>, ZPZV<2»; }; // NOLINT
                                                   template<> struct ConwayPolynomial<67, 5> { using ZPZ = aerobus::zpz<67>; using type =
 03502
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<65»; }; // NOLINT
                                                  template<> struct ConwayPolynomial<67, 6> { using ZPZ = aerobus::zpz<67>; using type =
03503
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<55>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<67, 7> { using ZPZ = aerobus::zpz<67>; using type
 03504
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<5»; }; // NOLINT
 03505
                                                 template<> struct ConwayPolynomial<67, 8> { using ZPZ = aerobus::zpz<67>; using type =
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<46>, ZPZV<17>, ZPZV<64>, ZPZV<64>; };
                                NOLINT
                               template<> struct ConwayPolynomial<67, 9> { using ZPZ = aerobus::zpz<67>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<49>, ZPZV<55>, ZPZV<65»; };
03506
                                                   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<16>, ZPZV<7>, ZPZV<23>,
                                ZPZV<2»; }; // NOLINT</pre>
                               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</pre>
03508
                                                  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<22>, ZPZV<21>, ZPZV<2
03510
                                                  template<> struct ConwayPolynomial<67, 13> { using ZPZ = aerobus::zpz<67>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2
                               ZPZV<56>, ZPZV<0>, ZPZV<1>, ZPZV<37>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<67, 15> { using ZPZ = aerobus::zpz<67>; using type =
03512
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<52>, ZPZV<41>, ZPZV<20>, ZPZV<21>, ZPZV<65»; }; // NOLINT
                                                 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<5>, ZPZV<65»; }; // NOLINT</pre>
                               template<> struct ConwayPolynomial
(67, 121
(7) Notifit
template
struct ConwayPolynomial
(7) Notifit
(87) Laves
(121
(87) Laves
(
03514
                               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<18>, ZPZV<65»; }; //</pre>
                                NOLINT
                                                   template<> struct ConwayPolynomial<71, 1> { using ZPZ = aerobus::zpz<71>; using type =
03516
                               POLYV<ZPZV<1>, ZPZV<64»: }; // NOLINT
                                                     template<> struct ConwayPolynomial<71, 2> { using ZPZ = aerobus::zpz<71>; using type =
                               POLYV<ZPZV<1>, ZPZV<69>, ZPZV<7»; }; // NOLINT
 03518
                                                 template<> struct ConwayPolynomial<71, 3> { using ZPZ = aerobus::zpz<71>; using type =
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<64»; }; // NOLINT template<> struct ConwayPolynomial<71, 4> { using ZPZ = aerobus::zpz<71>; using type =
 03519
                              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<41>, ZPZV<7»; }; // NOLINT 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
                                                  template<> struct ConwayPolynomial<71, 6> { using ZPZ = aerobus::zpz<71>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<13>, ZPZV<29>, ZPZV<7»; }; // NOLINT
                               template<> struct ConwayPolynomial<71, 7> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>; using type = emplate<> struct ConwayPolynomial<71, 8> { using ZPZ = aerobus::zpz<71>; using type =
 03522
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<22>, ZPZV<19>, ZPZV<7»; };
                                NOLINT
03524
                                                 template<> struct ConwayPolynomial<71, 9> { using ZPZ = aerobus::zpz<71>; using type =
                               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<43>, ZPZV<62>, ZPZV<64>; };
                                 // NOLINT
```

```
template<> struct ConwayPolynomial<71, 10> { using ZPZ = aerobus::zpz<71>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<53>, ZPZV<17>, ZPZV<26>, ZPZV<15, ZPZV<40>,
                                  ZPZV<7»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<71, 11> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
 03526
                                                                                                                                                                      // NOLINT
                                  ZPZV<48>, ZPZV<64»; };
                                 template<> struct ConwayPolynomial<71, 12> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<28>, ZPZV<29>, ZPZV<21>,
                                   ZPZV<58>, ZPZV<23>, ZPZV<7»; }; // NOLINT</pre>
                                 template<> struct ConwayPolynomial<71, 13> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03528
03529
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<32>, ZPZV<18>, ZPZV<52>, ZPZV<67>, ZPZV<49>, ZPZV<64»; }; // NOLINT</pre>
                                 \label{eq:convayPolynomial} $$ $$ template<> struct ConwayPolynomial<71, 17> { using ZPZ = aerobus::zpz<71>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03530
                                 ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZ
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<0>, ZPZV<0</pre>
                                  NOLINT
03532
                                                       template<> struct ConwayPolynomial<73, 1> { using ZPZ = aerobus::zpz<73>; using type =
                                 POLYV<ZPZV<1>, ZPZV<68»; }; // NOLINT
03533
                                                        template<> struct ConwayPolynomial<73, 2> { using ZPZ = aerobus::zpz<73>; using type =
                                  POLYV<ZPZV<1>, ZPZV<70>, ZPZV<5»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<73, 3> { using ZPZ = aerobus::zpz<73>; using type =
 03534
                                 POLYY<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<68»; }; // NOLINT template<> struct ConwayPolynomial<73, 4> { using ZPZ = aerobus::zpz<73>; using type =
 03535
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<56>, ZPZV<5»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<73, 5> { using ZPZ = aerobus::zpz<73>; using type =
03536
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<68»; }; // NOLINT
                                                        template<> struct ConwayPolynomial<73, 6> { using ZPZ = aerobus::zpz<73>; using type =
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<23>, ZPZV<48>, ZPZV<5»; }; // NOLINT
                               template<> struct ConwayPolynomial<73, 7> { using ZPZ = aerobus::zpz<73>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<68»; }; // NOLINT</pre>
 03538
                                                      template<> struct ConwayPolynomial<73, 8> { using ZPZ = aerobus::zpz<73>; using type =
03539
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<39>, ZPZV<18>, ZPZV<5»; };
                                 NOLINT
                                                      template<> struct ConwayPolynomial<73, 9> { using ZPZ = aerobus::zpz<73>; using type =
03540
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<72>, ZPZV<72>, ZPZV<15>, ZPZV<68»; };
                                   // NOLINT
                                                      template<> struct ConwayPolynomial<73, 10> { using ZPZ = aerobus::zpz<73>; using type =
03541
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<23>, ZPZV<33>, ZPZV<33>, ZPZV<36>,
                                  ZPZV<5»; }; // NOLINT</pre>
                                                     template<> struct ConwayPolynomial<73, 11> { using ZPZ = aerobus::zpz<73>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                 ZPZV<5>, ZPZV<68»; }; // NOLINT
    template<> struct ConwayPolynomial<73, 12> { using ZPZ = aerobus::zpz<73>; using type =
03543
                                 POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<52>, ZPZV<26>, ZPZV<26>, ZPZV<46>, ZPZV<46>, ZPZV<29>, ZPZV<25>, ZPZV<26>, ZPZV<26 , ZPZV<27 , ZPZV<27 , ZPZV<28 , ZPZV<
                                                       template<> struct ConwayPolynomial<73, 13> { using ZPZ = aerobus::zpz<73>; using type =
                                  \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                                 ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<68»; }; // NOLINT
template<> struct ConwayPolynomial<73, 15> { using ZPZ = aerobus::zpz<73>; using type =
03545
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
                                                     template<> struct ConwayPolynomial<73, 17> { using ZPZ = aerobus::zpz<73>; using type =
                                  POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                  ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<68»; }; // NOLINT</pre>
                                                       template<> struct ConwayPolynomial<73, 19> { using ZPZ = aerobus::zpz<73>; using type =
03547
                                  POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                   ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<68»; }; //</pre>
                                                       template<> struct ConwayPolynomial<79, 1> { using ZPZ = aerobus::zpz<79>; using type =
                                 POLYV<ZPZV<1>, ZPZV<76»; }; // NOLINT
 03549
                                                        template<> struct ConwayPolynomial<79, 2> { using ZPZ = aerobus::zpz<79>; using type =
                                 POLYV<ZPZV<1>, ZPZV<78>, ZPZV<3»; }; // NOLINT
                                                       template<> struct ConwayPolynomial<79, 3> { using ZPZ = aerobus::zpz<79>; using type =
03550
                                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<76»; }; // NOLINT template<> struct ConwayPolynomial<79, 4> { using ZPZ = aerobus::zpz<79>; using type =
 03551
                                 \label{eq:polyv} \mbox{PDLYV}<\mbox{ZPZV}<\mbox{1>, ZPZV}<\mbox{0>, ZPZV}<\mbox{2>, ZPZV}<\mbox{66>, ZPZV}<\mbox{3»; }; // \mbox{NOLINT}
                                 template<> struct ConwayPolynomial<79, 5> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<76»; }; // NOLINT
 03552
                                                        template<> struct ConwayPolynomial<79, 6> { using ZPZ = aerobus::zpz<79>; using type =
03553
                                 POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<28>, ZPZV<68>, ZPZV<3»; }; // NOLINI
                                                      template<> struct ConwayPolynomial<79, 7> { using ZPZ = aerobus::zpz<79>; using type
 03554
                                 POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<76»; }; // NOLINT
                                                      template<> struct ConwayPolynomial<79, 8> { using ZPZ = aerobus::zpz<79>; using type =
03555
                                  POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<60>, ZPZV<59>, ZPZV<48>, ZPZV<3»; };
                                  NOLINT
                                                       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<57>, ZPZV<19>, ZPZV<76»; };
                                   // NOLINT
03557
                                                      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
```

```
template<> struct ConwayPolynomial<79, 11> { using ZPZ = aerobus::zpz<79>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<3>, ZPZV<76»; }; // NOLINT</pre>
                           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>,
                                                                                                                                                                                // NOLINT
                            ZPZV<59>, ZPZV<62>, ZPZV<3»; };</pre>
                                               template<> struct ConwayPolynomial<79, 13> { using ZPZ = aerobus::zpz<79>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<78>, ZPZV<4>, ZPZV<76»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<br/>
779, 17> { using ZPZ = aerobus::zpz<79>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZP
03561
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<25>, ZPZV<76»; }; // NOLINT
03562
                                              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<25>, ZPZV<76»; }; //</pre>
                            NOLINT
03563
                                              template<> struct ConwayPolynomial<83, 1> { using ZPZ = aerobus::zpz<83>; using type =
                           POLYV<ZPZV<1>, ZPZV<81»; }; // NOLINT
                                               template<> struct ConwayPolynomial<83, 2> { using ZPZ = aerobus::zpz<83>; using type =
                           POLYV<ZPZV<1>, ZPZV<82>, ZPZV<2»; }; // NOLINT
                                               template<> struct ConwayPolynomial<83, 3> { using ZPZ = aerobus::zpz<83>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<81»; }; // NOLINT template<> struct ConwayPolynomial<83, 4> { using ZPZ = aerobus::zpz<83>; using type =
03566
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<42>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<83, 5> { using ZPZ = aerobus::zpz<83>; using type =
03567
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<81»; }; // NOLINT
                                            template<> struct ConwayPolynomial<83, 6> { using ZPZ = aerobus::zpz<83>; using type =
 03568
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<76>, ZPZV<32>, ZPZV<17>, ZPZV<2»; }; // NOLINT
                          template<> struct ConwayPolynomial<83, 7> { using ZPZ = aerobus::zpz<83>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<81»; }; // NOLINT</pre>
 03569
                                           template<> struct ConwayPolynomial<83, 8> { using ZPZ = aerobus::zpz<83>; using type =
 03570
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<23>, ZPZV<42>, ZPZV<2»; };
                                           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<24>, ZPZV<18>, ZPZV<81»; };
                            // NOLINT
                                              template<> struct ConwayPolynomial<83, 10> { using ZPZ = aerobus::zpz<83>; using type =
03572
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                            ZPZV<2»: }: // NOLINT
                                            template<> struct ConwayPolynomial<83, 11> { using ZPZ = aerobus::zpz<83>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<17>, ZPZV<81»; }; // NOLINT
                                              template<> struct ConwayPolynomial<83, 12> { using ZPZ = aerobus::zpz<83>; using type =
03574
                           POLYVCZPZVC1>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC0>, ZPZVC3>, ZPZVC12>, ZPZVC31>, ZPZVC19>, ZPZVC65>, ZPZVC55>, ZPZVC75>, ZPZVC75>, ZPZVC2»; }; // NOLINT
03575
                                           template<> struct ConwayPolynomial<83, 13> { using ZPZ = aerobus::zpz<83>; using type =
                           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03576
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<81»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<83, 19> { using ZPZ = aerobus::zpz<83>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<81»; }; //</pre>
                            NOLINT
03578
                                              template<> struct ConwayPolynomial<89, 1> { using ZPZ = aerobus::zpz<89>; using type =
                           POLYV<ZPZV<1>, ZPZV<86»; }; // NOLINT
                                           template<> struct ConwayPolynomial<89, 2> { using ZPZ = aerobus::zpz<89>; using type =
                           POLYV<ZPZV<1>, ZPZV<82>, ZPZV<3»; }; // NOLINT
 03580
                                               template<> struct ConwayPolynomial<89, 3> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<86»; }; // NOLINT template<> struct ConwayPolynomial<89, 4> { using ZPZ = aerobus::zpz<89>; using type =
03581
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<72>, ZPZV<3»; };
                                                                                                                                                                                                                                                                                        // NOLINT
                                               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
 03583
                                            template<> struct ConwayPolynomial<89, 6> { using ZPZ = aerobus::zpz<89>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<80>, ZPZV<15>, ZPZV<3»; }; // NOLINT
03584
                                            template<> struct ConwayPolynomial<89, 7> { using ZPZ = aerobus::zpz<89>; using type :
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<86»; }; // NOLINT
                                              template<> struct ConwayPolynomial<89, 8> { using ZPZ = aerobus::zpz<89>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<40>, ZPZV<79>, ZPZV<3»; };
                          template<> struct ConwayPolynomial<89, 9> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<12>, ZPZV<12>, ZPZV<66>, ZPZV<86»; };</pre>
03586
                             // NOLINT
                                              template<> struct ConwayPolynomial<89, 10> { using ZPZ = aerobus::zpz<89>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<16>, ZPZV<33>, ZPZV<82>, ZPZV<52>, ZPZV<4+,
                             ZPZV<3»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<89, 11> { using ZPZ = aerobus::zpz<89>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0 ,
                            ZPZV<26>, ZPZV<86»; }; // NOLINT</pre>
                                               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
03590
                                            template<> struct ConwayPolynomial<89, 13> { using ZPZ = aerobus::zpz<89>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
```

```
template<> struct ConwayPolynomial<89, 17> { using ZPZ = aerobus::zpz<89>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<86»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                                // NOLINT
                            template<> struct ConwayPolynomial<89, 19> { using ZPZ = aerobus::zpz<89>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZ
03592
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<34>, ZPZV<86»; }; //</pre>
                             NOLINT
 03593
                                                template<> struct ConwayPolynomial<97, 1> { using ZPZ = aerobus::zpz<97>; using type
                            POLYV<ZPZV<1>, ZPZV<92»; }; // NOLINT
                                             template<> struct ConwayPolynomial<97, 2> { using ZPZ = aerobus::zpz<97>; using type =
03594
                            POLYV<ZPZV<1>, ZPZV<96>, ZPZV<5»; }; // NOLINT
                                              template<> struct ConwayPolynomial<97, 3> { using ZPZ = aerobus::zpz<97>; using type =
 03595
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<92»; }; // NOLINT template<> struct ConwayPolynomial<97, 4> { using ZPZ = aerobus::zpz<97>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<80>, ZPZV<5»; }; // NOLINT
 03597
                                               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
                                               template<> struct ConwayPolynomial<97, 6> { using ZPZ = aerobus::zpz<97>; using type =
03598
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<92>, ZPZV<58>, ZPZV<58>, ZPZV<5»; }; // NOLINT
                                               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<0>, ZPZV<5>, ZPZV<92»; }; //
 03600
                                               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<55>, ZPZV<1>, ZPZV<32>, ZPZV<32>, ZPZV<5»; }; // NOLIN template<> struct ConwayPolynomial<97, 9> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<59>, ZPZV<7>, ZPZV<92»; };
                                                                                                                                                                                                                                                                                                                                                                                                                                                                        // NOLINT
03601
                             // NOLINT
                                             template<> struct ConwayPolynomial<97, 10> { using ZPZ = aerobus::zpz<97>; using type
03602
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<66>, ZPZV<34>, ZPZV<34>, ZPZV<34>, ZPZV<20>,
                             ZPZV<5»; }; // NOLINT</pre>
03603
                                             template<> struct ConwayPolynomial<97, 11> { using ZPZ = aerobus::zpz<97>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<5>, ZPZV<92»; };</pre>
                                                                                                                                       // NOLINT
                                               template<> struct ConwayPolynomial<97, 12> { using ZPZ = aerobus::zpz<97>; using type
03604
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<30>, ZPZV<59>, ZPZV<51>, ZPZV<81>, ZPZV<86>,
                             ZPZV<78>, ZPZV<94>, ZPZV<5»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<97, 13> { using ZPZ = aerobus::zpz<97>; using type =
03605
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                               template<> struct ConwayPolynomial<97, 17> { using ZPZ = aerobus::zpz<97>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             template<> struct ConwayPolynomial<97, 19> { using ZPZ = aerobus::zpz<97>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03607
                             ZPZV<0>, ZPZV<15>, ZPZV<92»; }; //</pre>
                             NOLINT
 03608
                                               template<> struct ConwayPolynomial<101, 1> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<99»; }; // NOLINT
03609
                                             template<> struct ConwayPolynomial<101, 2> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<97>, ZPZV<2»; }; // NOLINT
                                               template<> struct ConwayPolynomial<101, 3> { using ZPZ = aerobus::zpz<101>; using type =
 03610
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<99»; }; // NOLINT
                                                template<> struct ConwayPolynomial<101, 4> { using ZPZ = aerobus::zpz<101>; using type =
 03611
                            \label{eq:polyv} \mbox{PDLYV}<2\mbox{PZV}<1>, \mbox{ ZPZV}<0>, \mbox{ ZPZV}<1>, \mbox{ ZPZV}<7.8>, \mbox{ ZPZV}<2\mbox{ w; } \mbox{; } \mbox{/ NOLINT}
 03612
                                             template<> struct ConwayPolynomial<101, 5> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<99»; }; // NOLINT
                                               template<> struct ConwayPolynomial<101, 6> { using ZPZ = aerobus::zpz<101>; using type =
03613
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<90>, ZPZV<20>, ZPZV<67>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<101, 7> { using ZPZ = aerobus::zpz<101>; using type =
 03614
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<9»; }; // NOLINT
                            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<24>, ZPZV<29>, ZPZV<24>, ZPZV<29</pre>; }; //
 03615
                             NOLINT
03616
                                              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
03617
                                             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</pre>
                                             template<> struct ConwayPolynomial<101, 11> { using ZPZ = aerobus::zpz<101>; using type =
03618
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<31>, ZPZV<99»; }; // NOLINT</pre>
03619
                                            template<> struct ConwayPolynomial<101, 12> { using ZPZ = aerobus::zpz<101>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<79>, ZPZV<64>, ZPZV<39>, ZPZV<38>, ZPZV<48>, ZPZV<84>, ZPZV<84 , ZPZV<
                                               template<> struct ConwayPolynomial<101, 13> { using ZPZ = aerobus::zpz<101>; using type =
03620
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<99»; };</pre>
                                                                                                                                                                                                                          // NOLINT
03621
                                               template<> struct ConwayPolynomial<101, 17> { using ZPZ = aerobus::zpz<101>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<31>, ZPZV<99»; }; // NOLINT
template<> struct ConwayPolynomial<101, 19> { using ZPZ = aerobus::zpz<101>; using type
03622
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>
                             NOLINT
 03623
                                              template<> struct ConwayPolynomial<103, 1> { using ZPZ = aerobus::zpz<103>; using type =
                           POLYV<ZPZV<1>, ZPZV<98»; }; // NOLINT
                                             template<> struct ConwayPolynomial<103, 2> { using ZPZ = aerobus::zpz<103>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<102>, ZPZV<5»; };
                                        template<> struct ConwayPolynomial<103, 3> { using ZPZ = aerobus::zpz<103>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<98»; }; // NOLINT
                                       template<> struct ConwayPolynomial<103, 4> { using ZPZ = aerobus::zpz<103>; using type =
 03626
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<88>, ZPZV<5»; }; // NOLINT
                                        template<> struct ConwayPolynomial<103, 5> { using ZPZ = aerobus::zpz<103>; using type =
03627
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<98»; }; // NOLINT
                                         template<> struct ConwayPolynomial<103, 6> { using ZPZ = aerobus::zpz<103>; using type =
 03628
                         \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 9>, \ \texttt{ZPZV} < 30>, \ \texttt{ZPZV} < 5»; \ \}; \ \ // \ \ \texttt{NOLINT} 
 03629
                                      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<5-, ZPZV<5
                                     template<> struct ConwayPolynomial<103, 8> { using ZPZ = aerobus::zpz<103>; using type =
03630
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<71>, ZPZV<49>, ZPZV<5»; }; //
 03631
                                     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<97>, ZPZV<51>, ZPZV<98»; };
                        // NOLINT
03632
                                        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
                                       template<> struct ConwayPolynomial<103, 11> { using ZPZ = aerobus::zpz<103>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<5>, ZPZV<98»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<103, 12> { using ZPZ = aerobus::zpz<103>; using type =
03634
                        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
                                     template<> struct ConwayPolynomial<103, 13> { using ZPZ = aerobus::zpz<103>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<98»; };</pre>
                                                                                                                                                                                          // NOLINT
                                       template<> struct ConwayPolynomial<103, 17> { using ZPZ = aerobus::zpz<103>; using type =
03636
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<102>, ZPZV<8>, ZPZV<98»; );</pre>
                                                                                                                                                                                                                                                                                                                                             // NOLINT
                                       template<> struct ConwayPolynomial<103, 19> { using ZPZ = aerobus::zpz<103>; using type =
03637
                        POLYV<ZPZV<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<28»; }; //</pre>
                        NOLINT
03638
                                       template<> struct ConwayPolynomial<107, 1> { using ZPZ = aerobus::zpz<107>; using type =
                        POLYV<ZPZV<1>, ZPZV<105»; }; // NOLINT
                                         template<> struct ConwayPolynomial<107, 2> { using ZPZ = aerobus::zpz<107>; using type =
                        POLYV<ZPZV<1>, ZPZV<103>, ZPZV<2»; }; // NOLINT
03640
                                      template<> struct ConwayPolynomial<107, 3> { using ZPZ = aerobus::zpz<107>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<105»; ); // NOLINT
template<> struct ConwayPolynomial<107, 4> { using ZPZ = aerobus::zpz<107>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<79>, ZPZV<2»; }; // NOLINT
 03641
                                         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
 03643
                                      template<> struct ConwayPolynomial<107, 6> { using ZPZ = aerobus::zpz<107>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<5>, ZPZV<2>, ZPZV<29>, ZPZV<79>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<107, 7> { using ZPZ = aerobus::zpz<107>; using type =
03644
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<16>, ZPZV<105»; }; // NOLINT
                                        template<> struct ConwayPolynomial<107, 8> { using ZPZ = aerobus::zpz<107>; using type
 03645
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<24>, ZPZV<95>, ZPZV<2»; };
                        NOLINT
03646
                                       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
                                        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</pre>
03648
                                         template<> struct ConwayPolynomial<107, 11> { using ZPZ = aerobus::zpz<107>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                        template<> struct ConwayPolynomial<107, 12> { using ZPZ = aerobus::zpz<107>; using type
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<61>, ZPZV<42>, ZPZV<57>, ZPZV<57>, ZPZV<28; }; // NOLINT
03650
                                     template<> struct ConwayPolynomial<107, 13> { using ZPZ = aerobus::zpz<107>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<107, 17> { using ZPZ = aerobus::zpz<107>; using type =
03651
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<105»; };</pre>
03652
                                     template<> struct ConwayPolynomial<107, 19> { using ZPZ = aerobus::zpz<107>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        NOLINT
                                        template<> struct ConwayPolynomial<109, 1> { using ZPZ = aerobus::zpz<109>; using type =
                        POLYV<ZPZV<1>, ZPZV<103»; }; // NOLINT
                                      template<> struct ConwayPolynomial<109, 2> { using ZPZ = aerobus::zpz<109>; using type =
                       POLYV<ZPZV<1>, ZPZV<108>, ZPZV<6»; }; // NOLINT
                                       template<> struct ConwayPolynomial<109, 3> { using ZPZ = aerobus::zpz<109>; using type =
03655
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 4> { using ZPZ = aerobus::zpz<109>; using type =
 03656
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<98>, ZPZV<6»; }; // NOLINT
 03657
                                      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
                        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
 03658
```

```
template<> struct ConwayPolynomial<109, 7> { using ZPZ = aerobus::zpz<109>; using type
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT template<> struct ConwayPolynomial<109, 8> { using ZPZ = aerobus::zpz<109>; using type =
03660
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<34>, ZPZV<36>, ZPZV<6»; };
                       NOLINT
                                      template<> struct ConwayPolynomial<109, 9> { using ZPZ = aerobus::zpz<109>; using type =
03661
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<93>, ZPZV<87>, ZPZV<103»; };
                                      template<> struct ConwayPolynomial<109, 10> { using ZPZ = aerobus::zpz<109>; using type =
03662
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<71>, ZPZV<55>, ZPZV<16>, ZPZV<75>, ZPZV<69>,
                       ZPZV<6»; }; // NOLINT
                                      template<> struct ConwayPolynomial<109, 11> { using ZPZ = aerobus::zpz<109>; using type
03663
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<11>, ZPZV<103»; };
                                    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
03665
                                      template<> struct ConwayPolynomial<109, 13> { using ZPZ = aerobus::zpz<109>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; };</pre>
                                                                                                                                                                                            // NOLINT
                                      template<> struct ConwayPolynomial<109, 17> { using ZPZ = aerobus::zpz<109>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<103»; }; // NOLINT
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</pre>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>
03667
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<103»; }; //</pre>
03668
                                      template<> struct ConwayPolynomial<113, 1> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<110»; }; // NOLINT
                                      template<> struct ConwayPolynomial<113, 2> { using ZPZ = aerobus::zpz<113>; using type =
03669
                      POLYV<ZPZV<1>, ZPZV<101>, ZPZV<3»; }; // NOLINT
03670
                                        template<> struct ConwayPolynomial<113, 3> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<6>, ZPZV<10*,; }; // NOLINT template<> struct ConwayPolynomial<113, 4> { using ZPZ = aerobus::zpz<113>; using type =
03671
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<113, 5> { using ZPZ = aerobus::zpz<113>; using type =
03672
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<710»; }; // NOLINT 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
                                     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<5>, ZPZV<110»; }; // NOLINT
                                     template<> struct ConwayPolynomial<113, 8> { using ZPZ = aerobus::zpz<113>; using type =
03675
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<38>, ZPZV<38>, ZPZV<28>, ZPZV<28>, ZPZV<3»; }; //
                                       template<> struct ConwayPolynomial<113, 9> { using ZPZ = aerobus::zpz<113>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<87>, ZPZV<71>, ZPZV<110»; };
                                      template<> struct ConwayPolynomial<113, 10> { using ZPZ = aerobus::zpz<113>; using type :
03677
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<108>, ZPZV<57>, ZPZV<45>, ZPZV<45>, ZPZV<56>,
                       ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<113, 11> { using ZPZ = aerobus::zpz<113>; using type
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<3>, ZPZV<110»; }; // NOLINT</pre>
                       template<> struct ConwayPolynomial<113, 12> { using ZPZ = aerobus::zpz<113>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<62>, ZPZV<64>, ZPZV<48, ZPZV<56>,
03679
                       ZPZV<10>, ZPZV<27>, ZPZV<3»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<113, 13> { using ZPZ = aerobus::zpz<113>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<110»; }; // NOLINT</pre>
03681
                                        template<> struct ConwayPolynomial<113, 17> { using ZPZ = aerobus::zpz<113>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03682
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<110»; }; //</pre>
                       NOLINT
03683
                                       template<> struct ConwayPolynomial<127, 1> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<124»; }; // NOLINT
                                      template<> struct ConwayPolynomial<127, 2> { using ZPZ = aerobus::zpz<127>; using type =
03684
                       POLYV<ZPZV<1>, ZPZV<126>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<127, 3> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<124»; }; // NOLINT
                       template<> struct ConwayPolynomial<127, 4> { using ZPZ = aerobus::zpz<127>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<3>, ZPZV<3 , ZPZV<3 
03686
03687
                                       template<> struct ConwayPolynomial<127, 5> { using ZPZ = aerobus::zpz<127>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<124»; }; // NOLINT
                                      template<> struct ConwayPolynomial<127, 6> { using ZPZ = aerobus::zpz<127>; using type =
03688
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<84>, ZPZV<115>, ZPZV<82>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<127, 7> { using ZPZ = aerobus::zpz<127>; using type =
03689
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<124»; }; // NOLINT template<> struct ConwayPolynomial<127, 8> { using ZPZ = aerobus::zpz<127>; using type =
03690
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<104>, ZPZV<55>, ZPZV<8>, ZPZV<3»; };
03691
                                      template<> struct ConwayPolynomial<127, 9> { using ZPZ = aerobus::zpz<127>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<119>, ZPZV<126>, ZPZV<124»;
                       }; // NOLINT
03692
                                    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
                                         template<> struct ConwayPolynomial<127, 11> { using ZPZ = aerobus::zpz<127>; using type =
                           \texttt{POLYV} < \texttt{ZPZV} < \texttt{0} >, \ \texttt{ZPZV} < \texttt{
                           ZPZV<11>, ZPZV<124»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<127, 12> { using ZPZ = aerobus::zpz<127>; using type
03694
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<119>, ZPZV<25>, ZPZV<33>, ZPZV<97>, ZPZV<15>,
                                                                                                                                                                        // NOLINT
                           ZPZV<99>, ZPZV<8>, ZPZV<3»; };</pre>
                                          template<> struct ConwayPolynomial<127, 13> { using ZPZ = aerobus::zpz<127>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<127, 17> { using ZPZ = aerobus::zpz<127>; using type
03696
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<124»; };</pre>
                                         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>,
                                                                                                                                                                                                                                                                                                                                                                                                                             7.P7.V<0>.
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<30>, ZPZV<30>, ZPZV<30</pre>
                           NOLINT
                                            template<> struct ConwayPolynomial<131, 1> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<129»; }; // NOLINT
                                             template<> struct ConwayPolynomial<131, 2> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<127>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<131, 3> { using ZPZ = aerobus::zpz<131>; using type =
03700
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<129»; }; // NOLINT template<> struct ConwayPolynomial<131, 4> { using ZPZ = aerobus::zpz<131>; using type =
03701
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<109>, ZPZV<2»; }; // NOLINT
03702
                                           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
03703
                                            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 template<> struct ConwayPolynomial<131, 7> { using ZPZ = aerobus::zpz<131>; using type =
03704
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<1
                                            template<> struct ConwayPolynomial<131, 8> { using ZPZ = aerobus::zpz<131>; using type
03705
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<72>, ZPZV<116>, ZPZV<104>, ZPZV<2»; };
                           NOLINT
03706
                                           template<> struct ConwayPolynomial<131, 9> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<19>, ZPZV<129»; };
                                            template<> struct ConwayPolynomial<131, 10> { using ZPZ = aerobus::zpz<131>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<124>, ZPZV<97>, ZPZV<9>, ZPZV<126>, ZPZV<44>,
                           ZPZV<2»; }; // NOLINT</pre>
03708
                                           template<> struct ConwayPolynomial<131, 11> { using ZPZ = aerobus::zpz<131>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<6>, ZPZV<129»; };</pre>
                                                                                                                                     // NOLINT
                                            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<43>, ZPZV<125>,
                           ZPZV<28>, ZPZV<103>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<131, 13> { using ZPZ = aerobus::zpz<131>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<129»; };</pre>
                                                                                                                                                                                                                   // NOLINT
                                             template<> struct ConwayPolynomial<131, 17> { using ZPZ = aerobus::zpz<131>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                   // NOLINT
                            ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<129»; };</pre>
                          template<> struct ConwayPolynomial<131, 19> { using ZPZ = aerobus::zpz<131>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03712
                            ZPZV<0>, ZPZV<0>
                                           template<> struct ConwayPolynomial<137, 1> { using ZPZ = aerobus::zpz<137>; using type =
                          POLYV<ZPZV<1>, ZPZV<134»; }; // NOLINT
03714
                                             template<> struct ConwayPolynomial<137, 2> { using ZPZ = aerobus::zpz<137>; using type =
                         POLYV<ZPZV<1>, ZPZV<131>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<137, 3> { using ZPZ = aerobus::zpz<137>; using type =
03715
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<134»; }; // NOLINT
                                              template<> struct ConwayPolynomial<137, 4> { using ZPZ = aerobus::zpz<137>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<95>, ZPZV<3»; }; // NOLINT
03717
                                           template<> struct ConwayPolynomial<137, 5> { using ZPZ = aerobus::zpz<137>; using type =
                          template<> struct ConwayPolynomial<137, 6> { using ZPZ = aerobus::zpz<137>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<3>, ZPZV<3»; }; // NOLINT
03718
03719
                                             template<> struct ConwayPolynomial<137,
                                                                                                                                                                                                                              7> { using ZPZ = aerobus::zpz<137>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<13, ZPZV<134»; }; // NOLINT template<> struct ConwayPolynomial<137, 8> { using ZPZ = aerobus::zpz<137>; using type =
03720
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<105>, ZPZV<21>, ZPZV<34>, ZPZV<3*; };
                           NOLINT
                                           template<> struct ConwayPolynomial<137, 9> { using ZPZ = aerobus::zpz<137>; using type
03721
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<80>, ZPZV<122>, ZPZV<134»;
                           }; // NOLINT
03722
                                            template<> struct ConwayPolynomial<137, 10> { using ZPZ = aerobus::zpz<137>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<67>, ZPZV<93>, ZPZV<119>,
                           ZPZV<3»: }: // NOLINT
03723
                                             template<> struct ConwayPolynomial<137, 11> { using ZPZ = aerobus::zpz<137>; using type :
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<1>, ZPZV<134»; }; // NOLINT</pre>
03724
                                          template<> struct ConwayPolynomial<137, 12> { using ZPZ = aerobus::zpz<137>; using type
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<61>, ZPZV<40>, ZPZV<40>, ZPZV<36>, ZPZV<135>, ZPZV<61>, ZPZV<61>, ZPZV<33»; }; // NOLINT
03725
                                         template<> struct ConwayPolynomial<137, 13> { using ZPZ = aerobus::zpz<137>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<134»; }; // NOLINT
template<> struct ConwayPolynomial<137, 17> { using ZPZ = aerobus::zpz<137>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<136>, ZPZV<4>, ZPZV<134»; }; // NOLINT
template<> struct ConwayPolynomial<137, 19> { using ZPZ = aerobus::zpz<137>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                              ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<134»; }; //</pre>
                                               template<> struct ConwayPolynomial<139, 1> { using ZPZ = aerobus::zpz<139>; using type =
                             POLYV<ZPZV<1>, ZPZV<137»; }; // NOLINT
                                               template<> struct ConwayPolynomial<139, 2> { using ZPZ = aerobus::zpz<139>; using type =
                            POLYV<ZPZV<1>, ZPZV<138>, ZPZV<2»; }; // NOLINT
                                                 template<> struct ConwayPolynomial<139, 3> { using ZPZ = aerobus::zpz<139>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<137%; }; // NOLINT template<> struct ConwayPolynomial<139, 4> { using ZPZ = aerobus::zpz<139>; using type =
 03731
                           POLYV<2PZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<96>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<139, 5> { using ZPZ = aerobus:: ZpZ<139>; using type =
03732
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<137»; }; // NOLINT
                                               template<> struct ConwayPolynomial<139, 6> { using ZPZ = aerobus::zpz<139>; using type =
                             POLYV<2PZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<46>, ZPZV<10>, ZPZV<118>, ZPZV<2»; }; // NOLINT
 03734
                                              template<> struct ConwayPolynomial<139, 7> { using ZPZ = aerobus::zpz<139>; using type =
                            POLYV-ZPZV-1>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-0>, ZPZV-137»; }; / NOLINT template<> struct ConwayPolynomial<139, 8> { using ZPZ = aerobus::zpz<139>; using type =
                             POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<103>, ZPZV<36>, ZPZV<21>, ZPZV<22; };
                                             template<> struct ConwayPolynomial<139, 9> { using ZPZ = aerobus::zpz<139>; using type =
03736
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<70>, ZPZV<87>, ZPZV<87>, ZPZV<137»; };
                             // NOLINT
03737
                                               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»; };</pre>
                                                                                                                                              // NOLINT
                                               template<> struct ConwayPolynomial<139, 11> { using ZPZ = aerobus::zpz<139>; using type =
03738
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<7>, ZPZV<137»; }; // NOLINT</pre>
                             template<> struct ConwayPolynomial<139, 12> { using ZPZ = aerobus::zpz<139>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<120>, ZPZV<75>, ZPZV<41>, ZPZV<77>, ZPZV<77>, ZPZV<106>, ZPZV<8>, ZPZV<10>, ZPZV<2»; }; // NOLINT
03739
                                               template<> struct ConwayPolynomial<139, 13> { using ZPZ = aerobus::zpz<139>; using type
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<137»; }; // NOLINT
template<> struct ConwayPolynomial<139, 17> { using ZPZ = aerobus::zpz<139>; using type =
03741
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137»; };</pre>
                                                                                                                                                                                                                                                                                                                                                                                                                // NOLINT
                                               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<23>, ZPZV<137»; }; //</pre>
                             NOLINT
03743
                                               template<> struct ConwayPolynomial<149, 1> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<147»; }; // NOLINT
                                               template<> struct ConwayPolynomial<149, 2> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<145>, ZPZV<2»; }; // NOLINT
 03745
                                              template<> struct ConwayPolynomial<149, 3> { using ZPZ = aerobus::zpz<149>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<147»; }; // NOLINT template<> struct ConwayPolynomial<149, 4> { using ZPZ = aerobus::zpz<149>; using type =
03746
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<107>, ZPZV<2»; }; // NOLINT 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
 03748
                                               template<> struct ConwayPolynomial<149, 6> { using ZPZ = aerobus::zpz<149>; using type =
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<105>, ZPZV<33>, ZPZV<55>, ZPZV<2»; }; // NOLINT
03749
                                              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<1>, ZPZV<19>, ZPZV<147»; }; // NOLINT
                                               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<2w; };
03751
                                             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<0>, ZPZV<146>, ZPZV<146>, ZPZV<20>, ZPZV<147»;
                             }; // NOLINT
                                                 template<> struct ConwayPolynomial<149, 10> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<42>, ZPZV<148>, ZPZV<143>, ZPZV<51>,
                              ZPZV<2»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<149, 11> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<33>, ZPZV<147»; }; // NOLINT</pre>
                                                template<> struct ConwayPolynomial<149, 12> { using ZPZ = aerobus::zpz<149>; using type
03754
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<121>, ZPZV<91>, ZPZV<91>, ZPZV<52>, ZPZV<9>,
                             ZPZV<104>, ZPZV<110>, ZPZV<2»; }; // NOLINT</pre>
03755
                                              template<> struct ConwayPolynomial<149, 13> { using ZPZ = aerobus::zpz<149>; using type =
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                template<> struct ConwayPolynomial<149, 17> { using ZPZ = aerobus::zpz<149>; using type =
03756
                             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<447»; }; // NOLINT
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<5>, ZPZV<5>, ZPZV<147»; }; //
                             NOT.TNT
```

```
template<> struct ConwayPolynomial<151, 1> { using ZPZ = aerobus::zpz<151>; using type =
                      POLYV<ZPZV<1>, ZPZV<145»; }; // NOLINT
                                   template<> struct ConwayPolynomial<151, 2> { using ZPZ = aerobus::zpz<151>; using type =
                     POLYV<ZPZV<1>, ZPZV<149>, ZPZV<6»; }; // NOLINT
03760
                                    template<> struct ConwayPolynomial<151, 3> { using ZPZ = aerobus::zpz<151>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<145»; }; // NOLINT template<> struct ConwayPolynomial<151, 4> { using ZPZ = aerobus::zpz<151>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<89>, ZPZV<6»; }; // NOLINT
03762
                                   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
                                    template<> struct ConwayPolynomial<151, 6> { using ZPZ = aerobus::zpz<151>; using type =
03763
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<125>, ZPZV<18>, ZPZV<15>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<151, 7> { using ZPZ = aerobus::zpz<151>; using type
03764
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<145»; }; //
                                   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»; }; //
                      NOT.TNT
                      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<126>, ZPZV<126>, ZPZV<145»;
03766
                     }; // NOLINT
    template<> struct ConwayPolynomial<151, 10> { using ZPZ = aerobus::zpz<151>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<21>, ZPZV<104>, ZPZV<49>, ZPZV<20>, ZPZV<142>,
                       ZPZV<6»; }; // NOLINT
                                    template<> struct ConwayPolynomial<151, 11> { using ZPZ = aerobus::zpz<151>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<1>, ZPZV<145»; }; // NOLINT</pre>
                                   template<> struct ConwayPolynomial<151, 12> { using ZPZ = aerobus::zpz<151>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<109>, ZPZV<121>, ZPZV<101>, ZPZV<101>, ZPZV<6>, ZPZV<77>,
                       ZPZV<107>, ZPZV<147>, ZPZV<6»; }; // NOLINT</pre>
                                    template<> struct ConwayPolynomial<151, 13> { using ZPZ = aerobus::zpz<151>; using type =
03770
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<145»; }; // NOLINT</pre>
                                      template<> struct ConwayPolynomial<151, 17> { using ZPZ = aerobus::zpz<151>; using type =
03771
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<24>, ZPZV<145»; }; // NOLINT
template<> struct ConwayPolynomial<151, 19> { using ZPZ = aerobus::zpz<151>; using type =
03772
                       POLYY<ZPZV<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<9</pre>
                                      template<> struct ConwayPolynomial<157, 1> { using ZPZ = aerobus::zpz<157>; using type =
                     POLYV<ZPZV<1>, ZPZV<152»; }; // NOLINT
                                     template<> struct ConwayPolynomial<157, 2> { using ZPZ = aerobus::zpz<157>; using type =
03774
                      POLYV<ZPZV<1>, ZPZV<152>, ZPZV<5»: }: // NOLINT
                                    template<> struct ConwayPolynomial<157, 3> { using ZPZ = aerobus::zpz<157>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<152»; ); // NOLINT template<> struct ConwayPolynomial<157, 4> { using ZPZ = aerobus::zpz<157>; using type =
03776
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<136>, ZPZV<5»; }; // NOLINT
                                    template<> struct ConwayPolynomial<157, 5> { using ZPZ = aerobus::zpz<157>; using type =
03777
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<152»; }; // NOLINT
                                     template<> struct ConwayPolynomial<157, 6> { using ZPZ = aerobus::zpz<157>; using type =
03778
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<130>, ZPZV<43>, ZPZV<144>, ZPZV<5»; };
                                      template<> struct ConwayPolynomial<157, 7> { using ZPZ = aerobus::zpz<157>, using type =
                    POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<152»; };
03780
                                    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
                                      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
03782
                                      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</pre>
                                      template<> struct ConwayPolynomial<157, 11> { using ZPZ = aerobus::zpz<157>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<29>, ZPZV<152»; };</pre>
                                                                                                                 // NOLINT
03784
                                   template<> struct ConwayPolynomial<157, 12> { using ZPZ = aerobus::zpz<157>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, 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
                                    template<> struct ConwayPolynomial<157, 13> { using ZPZ = aerobus::zpz<157>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<156>, ZPZV<9>, ZPZV<152»; }; // NOLINT
template<> struct ConwayPolynomial<157, 17> { using ZPZ = aerobus::zpz<157>; using type =
03786
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      template<> struct ConwayPolynomial<157,
03787
                       POLYV<ZPZV<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<1>, ZPZV<1>, ZPZV<1</pre>; }; //
                                     template<> struct ConwayPolynomial<163, 1> { using ZPZ = aerobus::zpz<163>; using type =
                     POLYV<ZPZV<1>. ZPZV<161»: }: // NOLINT
                                      template<> struct ConwayPolynomial<163, 2> { using ZPZ = aerobus::zpz<163>; using type =
03789
                     POLYV<ZPZV<1>, ZPZV<159>, ZPZV<2»; }; // NOLINT
                                      template<> struct ConwayPolynomial<163, 3> { using ZPZ = aerobus::zpz<163>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<161»; };
                                                                                                                                                                                                             // NOLINT
                    template<> struct ConwayPolynomial<163, 4> { using ZPZ = aerobus::zpz<163>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<91>, ZPZV<2>; }; // NOLINT template<> struct ConwayPolynomial<163, 5> { using ZPZ = aerobus::zpz<163>; using type =
03791
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<161»; };
                                           template<> struct ConwayPolynomial<163, 6> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<83>, ZPZV<25>, ZPZV<156>, ZPZV<2»; }; // NOLINT
                                         template<> struct ConwayPolynomial<163, 7> { using ZPZ = aerobus::zpz<163>; using type =
03794
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<161»; }; // NOLINT template<> struct ConwayPolynomial<163, 8> { using ZPZ = aerobus::zpz<163>; using type =
03795
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<132>, ZPZV<83>, ZPZV<6>, ZPZV<6>, ZPZV<2»; }; //
                                        template<> struct ConwayPolynomial<163, 9> { using ZPZ = aerobus::zpz<163>; using type
03796
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<162>, ZPZV<161»;
                          }; // NOLINT
                                           template<> struct ConwayPolynomial<163, 10> { using ZPZ = aerobus::zpz<163>; using type =
03797
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<111>, ZPZV<120>, ZPZV<125>, ZPZV<15>, ZPZV<0>,
                           ZPZV<2»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<163, 11> { using ZPZ = aerobus::zpz<163>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<161»; }; // NOLINT
03799
                                         template<> struct ConwayPolynomial<163, 12> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<39>, ZPZV<112>, ZPZV<31>, ZPZV<38>, ZPZV<103>, ZPZV<103>, ZPZV<69>, ZPZV<2»; }; // NOLINT
                                           template<> struct ConwayPolynomial<163, 13> { using ZPZ = aerobus::zpz<163>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<161»; }; // NOLINT
template<> struct ConwayPolynomial<163, 17> { using ZPZ = aerobus::zpz<163>; using type =
03801
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<161»; }; // NOLINT</pre>
                                        template<> struct ConwayPolynomial<163,
                                                                                                                                                                                                                  19> { using ZPZ = aerobus::zpz<163>; using type =
03802
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<161»; }; //</pre>
                          NOLINT
                                         template<> struct ConwayPolynomial<167, 1> { using ZPZ = aerobus::zpz<167>; using type =
03803
                         POLYV<ZPZV<1>, ZPZV<162»; }; // NOLINT
                                          template<> struct ConwayPolynomial<167, 2> { using ZPZ = aerobus::zpz<167>; using type =
                         POLYV<ZPZV<1>, ZPZV<166>, ZPZV<5»; }; // NOLINT
                                           template<> struct ConwayPolynomial<167, 3> { using ZPZ = aerobus::zpz<167>; using type =
 03805
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162»; }; // NOLINT
template<> struct ConwayPolynomial<167, 4> { using ZPZ = aerobus::zpz<167>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<120>, ZPZV<5»; }; // NOLINT
03806
 03807
                                            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
 03808
                                        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 template<> struct ConwayPolynomial<167, 7> { using ZPZ = aerobus::zpz<167>; using type =
 03809
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10, ZPZV<102, ZPZV<102, ZPZV<102, ZPZV<102, ZPZV<103, ZPZV<103, ZPZV<104, ZPZV<10
                                           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»; };
03811
                                        template<> struct ConwayPolynomial<167, 9> { using ZPZ = aerobus::zpz<167>; using type
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<165>, ZPZV<165>, ZPZV<162»;
                          }; // NOLINT
03812
                                            template<> struct ConwayPolynomial<167, 10> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<85>, ZPZV<68>, ZPZV<109>, ZPZV<143>,
                           ZPZV<148>, ZPZV<5»; }; // NOLINT</pre>
03813
                                           template<> struct ConwayPolynomial<167, 11> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<24>, ZPZV<162»; ); // NOLINT
    template<> struct ConwayPolynomial<167, 12> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<142>, ZPZV<10>, ZPZV<142>, ZPZV<
                         ZPZV<140>, ZPZV<41>, ZPZV<57>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<167, 13> { using ZPZ = aerobus::zpz<167>; using type :
03815
                         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<162»; }; // NOLINT template<> struct ConwayPolynomial<167, 17> { using ZPZ = aerobus::zpz<167>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           \texttt{ZPZV} < \texttt{0>, ZPZV} < \texttt{2>, ZPZV} < \texttt{32>, ZPZV} < \texttt{162} ; }; 
                                                                                                                                                                                                                                                                                                                                                                   // NOLINT
                         template<> struct ConwayPolynomial<167, 19> { using ZPZ = aerobus::zpz<167>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
03817
03818
                                            template<> struct ConwayPolynomial<173, 1> { using ZPZ = aerobus::zpz<173>; using type =
                         POLYV<ZPZV<1>, ZPZV<171»; }; // NOLINT
 03819
                                        template<> struct ConwayPolynomial<173, 2> { using ZPZ = aerobus::zpz<173>; using type =
                         POLYV<ZPZV<1>, ZPZV<169>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<173, 3> { using ZPZ = aerobus::zpz<173>; using type =
 03820
                         POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 4> { using ZPZ = aerobus::zpz<173>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<102>, ZPZV<2»; }; // NOLINT
 03822
                                         template<> struct ConwayPolynomial<173, 5> { using ZPZ = aerobus::zpz<173>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<171»; }; // NOLINT
                                          template<> struct ConwayPolynomial<173, 6> { using ZPZ = aerobus::zpz<173>; using type =
03823
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<134>, ZPZV<107>, ZPZV<2»; }; // NOLINT
 03824
                                           template<> struct ConwayPolynomial<173,
                                                                                                                                                                                                                  7> { using ZPZ = aerobus::zpz<173>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5, ZP
 03825
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<125>, ZPZV<158>, ZPZV<27>, ZPZV<2»; }; //
                         NOLINT
 03826
                                        template<> struct ConwayPolynomial<173, 9> { using ZPZ = aerobus::zpz<173>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<104>, ZPZV<171»;
                            }; // NOLINT
03827
                                             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
template<> struct ConwayPolynomial<173, 11> { using ZPZ = aerobus::zpz<173>; using type =
03828
                            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<12>, ZPZV<171»; }; // NOLINT</pre>
                                            template<> struct ConwayPolynomial<173, 12> { using ZPZ = aerobus::zpz<173>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<2>, ZPZV<64>, ZPZV<46>, ZPZV<166>, ZPZV<10>,
                            ZPZV<159>, ZPZV<22>, ZPZV<2»; }; // NOLINT
                                             template<> struct ConwayPolynomial<173, 13> { using ZPZ = aerobus::zpz<173>; using type =
03830
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<173, 17> { using ZPZ = aerobus::zpz<173>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<171»; }; // NOLINT template<> struct ConwayPolynomial<173, 19> { using ZPZ = aerobus::zpz<173>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZ
03832
                             ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6</pre>
03833
                                              template<> struct ConwayPolynomial<179, 1> { using ZPZ = aerobus::zpz<179>; using type =
                           POLYV<ZPZV<1>, ZPZV<177»; }; // NOLINT
                                             template<> struct ConwayPolynomial<179, 2> { using ZPZ = aerobus::zpz<179>; using type =
03834
                           POLYV<ZPZV<1>, ZPZV<172>, ZPZV<2»; }; // NOLINT
                                               template<> struct ConwayPolynomial1779, 3> { using ZPZ = aerobus::zpz<179>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<177»; }; // NOLINT template<> struct ConwayPolynomial<179, 4> { using ZPZ = aerobus::zpz<179>; using type =
03836
                           template<> struct ConwayPolynomial<179, 5> { using ZPZ = aerobus::zpz<179>; using type =
03837
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<177»; }; // NOLINT
03838
                                               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
03839
                                           template<> struct ConwayPolynomial<179, 7> { using ZPZ = aerobus::zpz<179>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<177»; }; // NOLINT template<> struct ConwayPolynomial<179, 8> { using ZPZ = aerobus::zpz<179>; using type =
03840
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<144>, ZPZV<73>, ZPZV<2»; }; //
                                              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<40
                            // NOLINT
                           \label{eq:convergence} $$ \text{template} <> \text{struct ConwayPolynomial} <179, 10> { using ZPZ = $$ 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 <10>, ZPZV 
03842
                            ZPZV<2»; }; // NOLINT</pre>
                                              template<> struct ConwayPolynomial<179, 11> { using ZPZ = aerobus::zpz<179>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<28>, ZPZV<177»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<179, 12> { using ZPZ = aerobus::zpz<179>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<103>, ZPZV<83>, ZPZV<43>, ZPZV<76>, ZPZV<8>, ZPZV<177>, ZPZV<1>, ZPZV<1>, ZPZV<2»; }; // NOLINT
03844
03845
                                              template<> struct ConwayPolynomial<179, 13> { using ZPZ = aerobus::zpz<179>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                            ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<177»; }; // NOLINT</pre>
                                             template<> struct ConwayPolynomial<179, 17> { using ZPZ = aerobus::zpz<179>; using type =
03846
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4777; }; // NOLINT
template<> struct ConwayPolynomial<179, 19> { using ZPZ = aerobus::zpz<179>; using type
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                             ZPZV<0>, ZPZV<11>, ZPZV<17**, }; //</pre>
                            NOLINT
03848
                                             template<> struct ConwayPolynomial<181, 1> { using ZPZ = aerobus::zpz<181>; using type =
                           POLYV<ZPZV<1>, ZPZV<179»; }; // NOLINT
03849
                                               template<> struct ConwayPolynomial<181, 2> { using ZPZ = aerobus::zpz<181>; using type =
                           POLYV<ZPZV<1>, ZPZV<177>, ZPZV<2»; }; // NOLINT
03850
                                            template<> struct ConwayPolynomial<181, 3> { using ZPZ = aerobus::zpz<181>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<179»; }; // NOLINT
template<> struct ConwayPolynomial<181, 4> { using ZPZ = aerobus::zpz<181>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<105>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<181, 5> { using ZPZ = aerobus::zpz<181>; using type =
03851
03852
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<179»; }; // NOLINT
                                              template<> struct ConwayPolynomial<181, 6> { using ZPZ = aerobus::zpz<181>; using type =
03853
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<177>, ZPZV<163>, ZPZV<169>, ZPZV<2»; }; // NOLINT
03854
                                              template<> struct ConwayPolynomial<181, 7> { using ZPZ = aerobus::zpz<181>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179»; }; // NOLINT template<> struct ConwayPolynomial<181, 8> { using ZPZ = aerobus::zpz<181>; using type =
03855
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<108>, ZPZV<22>, ZPZV<149>, ZPZV<2); };
03856
                                             template<> struct ConwayPolynomial<181, 9> { using ZPZ = aerobus::zpz<181>; using type =
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<11>, ZPZV<107>, ZPZV<168>, ZPZV<179»;
                            }; // NOLINT
                                               template<> struct ConwayPolynomial<181, 10> { using ZPZ = aerobus::zpz<181>; using type =
03857
                            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<154>, ZPZV<104>, ZPZV<94>, ZPZV<57>, ZPZV<88>,
                            ZPZV<2»; }; // NOLINT</pre>
03858
                                            template<> struct ConwayPolynomial<181, 11> { using ZPZ = aerobus::zpz<181>; using type
                           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
03859
                                            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
template<> struct ConwayPolynomial<181, 13> { using ZPZ = aerobus::zpz<181>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<179»; };</pre>
                                                                                                                                                                                       // NOLINT
                                      template<> struct ConwayPolynomial<181, 17> { using ZPZ = aerobus::zpz<181>; using type =
03861
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<179»; };</pre>
                                                                                                                                                                                                                                                                                                                                   // NOLINT
                                   template<> struct ConwayPolynomial<181, 19> { using ZPZ = aerobus::zpz<181>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       NOLINT
03863
                                      template<> struct ConwayPolynomial<191, 1> { using ZPZ = aerobus::zpz<191>; using type =
                       POLYV<ZPZV<1>, ZPZV<172»; }; // NOLINT
                                    template<> struct ConwayPolynomial<191, 2> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<190>, ZPZV<19»; }; // NOLINT template<> struct ConwayPolynomial<191, 3> { using ZPZ = aerobus::zpz<191>; using type =
03865
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<172»; }; // NOLINT
                                      template<> struct ConwayPolynomial<191, 4> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<100>, ZPZV<19»; }; // NOLINT
                                       template<> struct ConwayPolynomial<191, 5> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<172»; }; // NOLINT
                                     template<> struct ConwayPolynomial<191, 6> { using ZPZ = aerobus::zpz<191>; using type =
03868
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<110>, ZPZV<10>, ZPZV<10>, ZPZV<10>, ZPZV<10>; y; // NOLINT template<> struct ConwayPolynomial<191, 7> { using ZPZ = aerobus::zpz<191>; using type
03869
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1+>, ZPZV<172»; }; // NOLINT
03870
                                     template<> struct ConwayPolynomial<191, 8> { using ZPZ = aerobus::zpz<191>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10+, ZPZV<164>, ZPZV<139>, ZPZV<171>, ZPZV<19w; }; //
                       NOLINT
03871
                                    template<> struct ConwayPolynomial<191, 9> { using ZPZ = aerobus::zpz<191>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<62>, ZPZV<62>, ZPZV<124>, ZPZV<172»;
                      }; // NOLINT
    template<>> struct ConwayPolynomial<191, 10> { using ZPZ = aerobus::zpz<191>; using type
03872
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<113>, ZPZV<47>, ZPZV<173>, ZPZV<74>,
                       ZPZV<156>, ZPZV<19»; }; // NOLINT
03873
                                      template<> struct ConwayPolynomial<191, 11> { using ZPZ = aerobus::zpz<191>; using type =
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                      template<> struct ConwayPolynomial<191, 12> { using ZPZ = aerobus::zpz<191>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<168>, ZPZV<25>, ZPZV<49>, ZPZV<90>,
                       ZPZV<7>, ZPZV<151>, ZPZV<19»; }; // NOLINT</pre>
                      \label{eq:convayPolynomial} $$$ template<> struct ConwayPolynomial<191, 13> { using ZPZ = aerobus::zpz<191>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>,
03875
                       ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<172»; }; // NOLINT
                                      template<> struct ConwayPolynomial<191, 17> { using ZPZ = aerobus::zpz<191>; using type
                      POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                       template<> struct ConwayPolynomial<191, 19> { using ZPZ = aerobus::zpz<191>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<190>, ZPZV<2>, ZPZV<172»; }; //
03877
                      NOLINT
                                       template<> struct ConwayPolynomial<193, 1> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYY<ZPZV<1>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 2> { using ZPZ = aerobus::zpz<193>; using type =
03879
                     POLYV<ZPZV<1>, ZPZV<192>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<193, 3> { using ZPZ = aerobus::zpz<193>; using type =
03880
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<188»; }; // NOLINT
                                    template<> struct ConwayPolynomial<193, 4> { using ZPZ = aerobus::zpz<193>; using type =
03881
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<48>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<193, 5> { using ZPZ = aerobus::zpz<193>; using type =
03882
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<188»; }; // NOLINT template<> struct ConwayPolynomial<193, 6> { using ZPZ = aerobus::zpz<193>; using type =
03883
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<149>, ZPZV<8>, ZPZV<172>, ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<193, 7> { using ZPZ = aerobus::zpz<193>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<188»; };
                      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»; }; //</pre>
03885
                      NOLINT
                                     template<> struct ConwayPolynomial<193, 9> { using ZPZ = aerobus::zpz<193>; using type =
03886
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<168>, ZPZV<168>, ZPZV<27>, ZPZV<188»;
                       }; // NOLINT
03887
                                      template<> struct ConwayPolynomial<193, 10> { using ZPZ = aerobus::zpz<193>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<51>, ZPZV<77>, ZPZV<0>, ZPZV<89>,
                       ZPZV<5»; }; // NOLINT
                                      template<> struct ConwayPolynomial<193, 11> { using ZPZ = aerobus::zpz<193>; using type =
03888
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                  // NOLINT
                       ZPZV<1>, ZPZV<188»; };</pre>
03889
                                    template<> struct ConwayPolynomial<193, 12> { using ZPZ = aerobus::zpz<193>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<55>, ZPZV<52>, ZPZV<135>, ZPZV<155>, ZPZV<
                      ZPZV<90>, ZPZV<46>, ZPZV<28>, ZPZV<5»; }; // NOLINT
template<> struct ConwayPolynomial<193, 13> { using ZPZ = aerobus::zpz<193>; using type :
03890
                        POLYV<ZPZV<1>, ZPZV<0>, 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</pre>
                                    template<> struct ConwayPolynomial<193, 17> { using ZPZ = aerobus::zpz<193>; using type :
                      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
03892
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                           ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<188»; }; //</pre>
                          NOLINT
03893
                                          template<> struct ConwayPolynomial<197, 1> { using ZPZ = aerobus::zpz<197>; using type =
                          POLYV<ZPZV<1>, ZPZV<195»; }; // NOLINT
                                           template<> struct ConwayPolynomial<197, 2> { using ZPZ = aerobus::zpz<197>; using type =
03894
                          POLYV<ZPZV<1>, ZPZV<192>, ZPZV<2»; }; // NOLINT
                                            template<> struct ConwayPolynomial<197, 3> { using ZPZ = aerobus::zpz<197>; using type =
 03895
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<195»; }; // NOLINT template<> struct ConwayPolynomial<197, 4> { using ZPZ = aerobus::zpz<197>; using type =
 03896
                         POLYV-ZPZV-1>, ZPZV-(1>, ZPZV-16>, ZPZV-124>, ZPZV-2; }; // NOLINT template<> struct ConwayPolynomial<197, 5> { using ZPZ = aerobus::zpz<197>; using type =
 03897
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<195»; }; // NOLINT
                                            template<> struct ConwayPolynomial<197, 6> { using ZPZ = aerobus::zpz<197>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<124>, ZPZV<79>, ZPZV<173>, ZPZV<2»; }; // NOLINT
 03899
                                          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<0>, ZPZV<5, ZPZV<6>, ZPZV<6>, ZPZV<195»; }; // NOLINT template<> struct ConwayPolynomial<197, 8> { using ZPZ = aerobus::zpz<197>; using type =
03900
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<176>, ZPZV<96>, ZPZV<29>, ZPZV<2»; };
                          template<> struct ConwayPolynomial<197, 9> { using ZPZ = aerobus::zpz<197>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<127>, ZPZV<8>, ZPZV<195»;
                          }; // NOLINT
                          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>,
03902
                          ZPZV<2»; }; // NOLINT
                                        template<> struct ConwayPolynomial<197, 11> { using ZPZ = aerobus::zpz<197>; using type =
 03903
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<14>, ZPZV<195»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<197, 12> { using ZPZ = aerobus::zpz<197>; using type =
03904
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<168>, ZPZV<15>, ZPZV<130>, ZPZV<141>, ZPZV<9>,
                          ZPZV<90>, ZPZV<163>, ZPZV<2»; }; // NOLINT</pre>
                                           template<> struct ConwayPolynomial<197, 13> { using ZPZ = aerobus::zpz<197>; using type
03905
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<3>, ZPZV<39>, ZPZV<195»; }; // NOLINT
template<> struct ConwayPolynomial<197, 17> { using ZPZ = aerobus::zpz<197>; using type =
03906
                          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<195»; }; // NOLINT</pre>
 03907
                                            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<6>, ZPZV<6>, ZPZV<6>, ZPZV<6</pre>; }; //
                          NOLINT
03908
                                          template<> struct ConwayPolynomial<199, 1> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<196»; }; // NOLINT
                                            template<> struct ConwayPolynomial<199, 2> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<193>, ZPZV<3»; }; // NOLINT
 03910
                                         template<> struct ConwayPolynomial<199, 3> { using ZPZ = aerobus::zpz<199>; using type =
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<196»; }; // NOLINT template<> struct ConwayPolynomial<199, 4> { using ZPZ = aerobus::zpz<199>; using type =
03911
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<3»; }; // NOLINT
                                           template<> struct ConwayPolynomial<199, 5> { using ZPZ = aerobus::zpz<199>; using type =
03912
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
 03913
                                           template<> struct ConwayPolynomial<199, 6> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9\), ZPZV<5\), ZPZV<5\), ZPZV<5\), ZPZV<5\), ZPZV<5\), ZPZV<5\), ZPZV<5\), ZPZV<5\); y; // NOLINT template<> struct ConwayPolynomial<199, 7> { using ZPZ = aerobus::zpz<199>; using type =
03914
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<196»; }; // NOLINT
                                            template<> struct ConwayPolynomial<199, 8> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<160>, ZPZV<23>, ZPZV<159>, ZPZV<3»; };
03916
                                           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<87>, ZPZV<177>, ZPZV<141>, ZPZV<196»;
                          }; // NOLINT
03917
                                            template<> struct ConwayPolynomial<199, 10> { using ZPZ = aerobus::zpz<199>; using type =
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<171>, ZPZV<158>, ZPZV<31>, ZPZV<54>, ZPZV<9>,
                           ZPZV<3»; }; // NOLINT</pre>
03918
                                        template<> struct ConwayPolynomial<199, 11> { using ZPZ = aerobus::zpz<199>; using type =
                           \texttt{POLYV} < \texttt{ZPZV} < 1>, \quad \texttt{ZPZV} < 0>, \quad 
                          ZPZV<1>, ZPZV<196»; }; // NOLINT</pre>
                                          template<> struct ConwayPolynomial<199, 12> { using ZPZ = aerobus::zpz<199>; using type =
03919
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<33>, ZPZV<192>, ZPZV<197>, ZPZV<138>,
                          ZPZV<69>, ZPZV<57>, ZPZV<151>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<199, 13> { using ZPZ = aerobus::zpz<199>; using type =
03920
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                           template<> struct ConwayPolynomial<199,
                                                                                                                                                                                                                     17> { using ZPZ = aerobus::zpz<199>; using type =
03921
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                          ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<196»; }; // NOLINT</pre>
03922
                                          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<196»; }; //</pre>
                          NOLINT
03923
                                           template<> struct ConwayPolynomial<211, 1> { using ZPZ = aerobus::zpz<211>; using type =
                          POLYV<ZPZV<1>, ZPZV<209»; }; // NOLINT
 03924
                                         template<> struct ConwayPolynomial<211, 2> { using ZPZ = aerobus::zpz<211>; using type =
                          POLYV<ZPZV<1>, ZPZV<207>, ZPZV<2»; }; // NOLINT
                                           \texttt{template<> struct ConwayPolynomial<211, 3> \{ using ZPZ = aerobus:: zpz<211>; using type = aerob
 03925
                          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<209»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<211, 4> { using ZPZ = aerobus::zpz<211>; using type =
03926
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<61, ZPZV<2; }; // NOLINT
template<> struct ConwayPolynomial<211, 5> { using ZPZ = aerobus::zpz<211>; using type =
03927
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<209»; }; // NOLINT
                      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
 03928
                                       template<> struct ConwayPolynomial<211, 7> { using ZPZ = aerobus::zpz<211>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<209»; };
                                    template<> struct ConwayPolynomial<211, 8> { using ZPZ = aerobus::zpz<211>; using type =
 03930
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<200>, ZPZV<87>, ZPZV<29>, ZPZV<2»; };
                       NOLINT
                                     template<> struct ConwayPolynomial<211, 9> { using ZPZ = aerobus::zpz<211>; using type =
03931
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<139>, ZPZV<139>, ZPZV<26>, ZPZV<209»;
                       }; // NOLINT
                                    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»; }; // NOLTNT</pre>
                                     template<> struct ConwayPolynomial<211, 11> { using ZPZ = aerobus::zpz<211>; using type
03933
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<7>, ZPZV<209»; };</pre>
                                                                                                                   // NOLINT
                                     template<> struct ConwayPolynomial<211, 12> { using ZPZ = aerobus::zpz<211>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<50>, ZPZV<50>, ZPZV<145>, ZPZV<126>, ZPZV<184>,
                       ZPZV<84>, ZPZV<27>, ZPZV<2»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<211, 13> { using ZPZ = aerobus::zpz<211>; using type =
03935
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                                    template<> struct ConwayPolynomial<211,
                                                                                                                                                                                              17> { using ZPZ = aerobus::zpz<211>; using type =
03936
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<209»; }; // NOLINT
template<> struct ConwayPolynomial<211, 19> { using ZPZ = aerobus::zpz<211>; using type =
03937
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<1>, ZPZV<1</pre>
                                     template<> struct ConwayPolynomial<223, 1> { using ZPZ = aerobus::zpz<223>; using type =
03938
                       POLYV<ZPZV<1>, ZPZV<220»; }; // NOLINT
                                       template<> struct ConwayPolynomial<223, 2> { using ZPZ = aerobus::zpz<223>; using type =
03939
                       POLYV<ZPZV<1>, ZPZV<221>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<223, 3> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 4> { using ZPZ = aerobus::zpz<223>; using type =
 03941
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<163>, ZPZV<3»; }; // NOLINT
                                       template<> struct ConwayPolynomial<223, 5> { using ZPZ = aerobus::zpz<223>; using type =
03942
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<220»; }; // NOLINT
03943
                                       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
 03944
                                       template<> struct ConwayPolynomial<223, 7> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<220»; }; // NOLINT template<> struct ConwayPolynomial<223, 8> { using ZPZ = aerobus::zpz<223>; using type =
03945
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<139>, ZPZV<98>, ZPZV<138>, ZPZV<138>, ZPZV<5»; };
                       NOLINT
03946
                                       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<0>, ZPZV<1>, ZPZV<164>, ZPZV<64>, ZPZV<220»;
                       }; // NOLINT
03947
                                      template<> struct ConwayPolynomial<223, 10> { using ZPZ = aerobus::zpz<223>; using type =
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<177>, ZPZV<87>, ZPZV<99>, ZPZV<62>,
                       ZPZV<3»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<223, 11> { using ZPZ = aerobus::zpz<223>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<8>, ZPZV<220»; }; // NOLINT</pre>
03949
                                       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<213>, ZPZV<213>, ZPZV<3»; }; // NOLINT
03950
                                       template<> struct ConwayPolynomial<223, 13> { using ZPZ = aerobus::zpz<223>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<23>, ZPZV<220»; }; // NOLINT</pre>
03951
                                    template<> struct ConwayPolynomial<223, 17> { using ZPZ = aerobus::zpz<223>; using type =
                        \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ 
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; JPZV<0>; JPZV<0>; ZPZV<0>; JPZV<0>; JPZ
03952
                        POLYV<ZPZV<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<2>, ZPZV<2</pre>
03953
                                       template<> struct ConwayPolynomial<227, 1> { using ZPZ = aerobus::zpz<227>; using type =
                       POLYV<ZPZV<1>, ZPZV<225»; }; // NOLINT
                                       template<> struct ConwayPolynomial<227, 2> { using ZPZ = aerobus::zpz<227>; using type =
03954
                       POLYV<ZPZV<1>, ZPZV<220>, ZPZV<2»; }; // NOLINT
                                      template<> struct ConwayPolynomial<227, 3> { using ZPZ = aerobus::zpz<227>; using type =
 03955
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<225»; }; // NOLINT template<> struct ConwayPolynomial<227, 4> { using ZPZ = aerobus::zpz<227>; using type =
 03956
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<143>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<227, 5> { using ZPZ = aerobus::zpz<227>; using type =
 03957
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<225»; }; // NOLINT
                                       template<> struct ConwayPolynomial<227, 6> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<174>, ZPZV<24>, ZPZV<135>, ZPZV<2»; }; // NOLINT
                                    template<> struct ConwayPolynomial<227, 7> { using ZPZ = aerobus::zpz<227>; using type =
 03959
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>; Wing ZPZ = aerobus::2pz<227>; using type =
 03960
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<151>, ZPZV<176>, ZPZV<106>, ZPZV<2»; }; //
                                    template<> struct ConwayPolynomial<227, 9> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<24>, ZPZV<24>, ZPZV<183>, ZPZV<225»;
                       }; // NOLINT
                                       template<> struct ConwayPolynomial<227, 10> { using ZPZ = aerobus::zpz<227>; using type =
03962
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<22>, ZPZV<199>, ZPZV<12>, ZPZV<93>, ZPZV<77>,
                       ZPZV<2»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<227, 11> { using ZPZ = aerobus::zpz<227>; using type =
03963
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<2>, ZPZV<225»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<227, 12> { using ZPZ = aerobus::zpz<227>; using type =
03964
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<123>, ZPZV<99>, ZPZV<160>, ZPZV<96>, ZPZV<127>, ZPZV<142>, ZPZV<94>, ZPZV<2»; }; // NOLINT
03965
                                   template<> struct ConwayPolynomial<227, 13> { using ZPZ = aerobus::zpz<227>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                     template<> struct ConwayPolynomial<227, 17> { using ZPZ = aerobus::zpz<227>; using type =
03966
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<225»; };</pre>
                                     template<> struct ConwayPolynomial<227, 19> { using ZPZ = aerobus::zpz<227>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<34>, ZPZV<25»; }; //</pre>
                       NOLINT
03968
                                      template<> struct ConwayPolynomial<229, 1> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<223»; }; // NOLINT
                                     template<> struct ConwayPolynomial<229, 2> { using ZPZ = aerobus::zpz<229>; using type =
03969
                      POLYV<ZPZV<1>, ZPZV<228>, ZPZV<6»; }; // NOLINT
03970
                                      template<> struct ConwayPolynomial<229, 3> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<223»; }; // NOLINT
template<> struct ConwayPolynomial<2229, 4> { using ZPZ = aerobus::zpz<229>; using type =
03971
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<162>, ZPZV<6»; };
                                                                                                                                                                                                                                                // NOLINT
                                      template<> struct ConwayPolynomial<229, 5> { using ZPZ = aerobus::zpz<229>; using type =
03972
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223»; }; // NOLINT
03973
                                      template<> struct ConwayPolynomial<229, 6> { using ZPZ = aerobus::zpz<229>; using type =
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<24>, ZPZV<160>, ZPZV<186>, ZPZV<6»; }; // NOLINT
                                     template<> struct ConwayPolynomial<229, 7> { using ZPZ = aerobus::zpz<229>; using type =
03974
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<23»; }; //
                                      template<> struct ConwayPolynomial<229, 8> { using ZPZ = aerobus::zpz<229>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<193>, ZPZV<62>, ZPZV<205>, ZPZV<60; };
                       NOLINT
03976
                     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»;</pre>
                     }; // NOLINT
    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>,
                                     template<> struct ConwayPolynomial<229, 11> { using ZPZ = aerobus::zpz<229>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<223»; }; // NOLINT
                                      template<> struct ConwayPolynomial<229, 12> { using ZPZ = aerobus::zpz<229>; using type
03979
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<131>, ZPZV<140>, ZPZV<25>, ZPZV<6>, ZPZV<172>, ZPZV<9>, ZPZV<145>, ZPZV<6»; }; // NOLINT
03980
                                      template<> struct ConwayPolynomial<229, 13> { using ZPZ = aerobus::zpz<229>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<47>, ZPZV<223»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<229, 17> { using ZPZ = aerobus::zpz<229>; using type
                       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<223»; };</pre>
                                                                                                                                                                                                                                                                                                                               // NOLINT
03982
                                       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<2>, ZPZV<2>, ZPZV<223»; }; //</pre>
                       NOLINT
                                       template<> struct ConwayPolynomial<233, 1> { using ZPZ = aerobus::zpz<233>; using type =
                      POLYV<ZPZV<1>, ZPZV<230»; }; // NOLINT
03984
                                     template<> struct ConwayPolynomial<233, 2> { using ZPZ = aerobus::zpz<233>; using type =
                      POLYV<ZPZV<1>, ZPZV<232>, ZPZV<3»; }; // NOLINT
                                     template<> struct ConwayPolynomial<233, 3> { using ZPZ = aerobus::zpz<233>; using type =
03985
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<230»; }; // NOLINT
                                       template<> struct ConwayPolynomial<233, 4> { using ZPZ = aerobus::zpz<233>; using type =
03986
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<158>, ZPZV<3»; }; // NOLINT
                                   template<> struct ConwayPolynomial<233, 5> { using ZPZ = aerobus::zpz<233>; using type =
03987
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<230»; }; // NOLINT template<> struct ConwayPolynomial<233, 6> { using ZPZ = aerobus::zpz<233>; using type =
03988
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<122>, ZPZV<32>, ZPZV<32>, ZPZV<32>, ZPZV<32>; // NOLINT template<> struct ConwayPolynomial<233, 7> { using ZPZ = aerobus::zpz<233>; using type
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<230»; }; //
03990
                                    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
                                     template<> struct ConwayPolynomial<233, 9> { using ZPZ = aerobus::zpz<233>; using type
03991
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<56>, ZPZV<146>, ZPZV<230»;
                       }; // NOLINT
03992
                                     template<> struct ConwayPolynomial<233, 10> { using ZPZ = aerobus::zpz<233>; using type
                       \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 28>, \ \texttt{ZPZV} < 71>, \ \texttt{ZPZV} < 102>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 48>, \ \texttt{ZPZV} < 102>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 102>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 102>, \ \texttt{ZPZV} < 3>, \ \texttt{ZPZV} < 102>, \ \texttt
                       ZPZV<3»; }; // NOLINT</pre>
03993
                                    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<0>, ZPZV<0>,
                         ZPZV<5>, ZPZV<230»; }; // NOLINT</pre>
                                       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</pre>
                                         template<> struct ConwayPolynomial<233, 13> { using ZPZ = aerobus::zpz<233>; using type =
03995
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<230»; }; // NOLINT</pre>
                                       template<> struct ConwayPolynomial<233,
                                                                                                                                                                                                             17> { using ZPZ = aerobus::zpz<233>; using type =
03996
                        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<23); // NOLINT template<> struct ConwayPolynomial<233, 19> { using ZPZ = aerobus::zpz<233>; using type =
03997
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                                                                                              ZPZV<0>.
                         ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<25>, ZPZV<25>, ZPZV<230»; }; //</pre>
                         NOLINT
03998
                                         template<> struct ConwayPolynomial<239, 1> { using ZPZ = aerobus::zpz<239>; using type =
                       POLYV<ZPZV<1>, ZPZV<232»; }; // NOLINT
                                         template<> struct ConwayPolynomial<239, 2> { using ZPZ = aerobus::zpz<239>; using type =
03999
                         POLYV<ZPZV<1>, ZPZV<237>, ZPZV<7»; }; // NOLINT
                                        template<> struct ConwayPolynomial<239, 3> { using ZPZ = aerobus::zpz<239>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<232»; }; // NOLINT template<> struct ConwayPolynomial<239, 4> { using ZPZ = aerobus::zpz<239>; using type =
04001
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<132>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<239, 5> { using ZPZ = aerobus::zpz<239>; using type =
04002
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; }; // NOLINT
                                          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
04004
                                        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<0>, ZPZV<17>, ZPZV<232»; };
04005
                                        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
                                        template<> struct ConwayPolynomial<239, 9> { using ZPZ = aerobus::zpz<239>; using type =
04006
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<2>, ZPZV<88>, ZPZV<3232»; };
                         // NOLINT
                                        template<> struct ConwayPolynomial<239, 10> { using ZPZ = aerobus::zpz<239>; using type =
04007
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<57>, ZPZV<68>, ZPZV<226>, ZPZV<127>, ZPZV<108>, ZPZV<7»; }; // NOLINT
                                          template<> struct ConwayPolynomial<239, 11> { using ZPZ = aerobus::zpz<239>; using type
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<8>, ZPZV<232»; }; // NOLINT</pre>
                       template<>> struct ConwayPolynomial<239, 12> { using ZPZ = aerobus::zpz<239>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<235>, ZPZV<14>, ZPZV<14>, ZPZV<113>, ZPZV<182>,
ZPZV<101>, ZPZV<81>, ZPZV<216>, ZPZV<7»; }; // NOLINT
template<>> struct ConwayPolynomial<239, 13> { using ZPZ = aerobus::zpz<239>; using type =
04009
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<232»; };</pre>
                                                                                                                                                                                                 // NOLINT
                        template<> struct ConwayPolynomial<239, 17> { using ZPZ = aerobus::zpz<239>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04011
                          \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 232*; \ \}; \ \ // \ \texttt{NOLINT} 
                                         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<24>, ZPZV<24>, ZPZV<232»; }; //</pre>
                         NOLINT
04013
                                        \texttt{template} <> \texttt{struct ConwayPolynomial} < 241, 1 > \{ \texttt{using ZPZ = aerobus:: zpz} < 241 >; \texttt{using type = aerobus:: zpz} < 241 >; using type = aerobus: zpz < 241 >; using type = aerobus: 
                        POLYV<ZPZV<1>, ZPZV<234»; }; // NOLINT
                                          template<> struct ConwayPolynomial<241, 2> { using ZPZ = aerobus::zpz<241>; using type =
                        POLYV<ZPZV<1>, ZPZV<238>, ZPZV<7»; }; // NOLINT
                                          template<> struct ConwayPolynomial<241, 3> { using ZPZ = aerobus::zpz<241>; using type =
04015
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<234»; }; // NOLINT template<> struct ConwayPolynomial<241, 4> { using ZPZ = aerobus::zpz<241>; using type =
04016
                       POLYV<2PZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<152, ZPZV<18; }; // NOLINT template<> struct ConwayPolynomial<241, 5> { using ZPZ = aerobus::zpz<241>; using type =
04017
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<234»; }; // NOLINT
04018
                                       template<> struct ConwayPolynomial<241, 6> { using ZPZ = aerobus::zpz<241>; using type =
                        template<> struct ConwayPolynomial<241, 7> { using ZPZ = aerobus::zpz<241>; using type =
04019
                        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<234»; }; // NOLINT
                                       template<> struct ConwayPolynomial<241, 8> { using ZPZ = aerobus::zpz<241>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<173>, ZPZV<212>, ZPZV<153>, ZPZV<7»; }; //
04021
                                      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<0>, ZPZV<236>, ZPZV<125>, ZPZV<234»;
                         }; // NOLINT
                                          template<> struct ConwayPolynomial<241, 10> { using ZPZ = aerobus::zpz<241>; using type =
04022
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<29>, ZPZV<27>, ZPZV<145>, ZPZV<268>, ZPZV<55>,
                         ZPZV<7»; }; // NOLINT</pre>
04023
                                        template<> struct ConwayPolynomial<241, 11> { using ZPZ = aerobus::zpz<241>; using type =
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                         ZPZV<3>, ZPZV<234»; }; // NOLINT</pre>
                                         template<> struct ConwayPolynomial<241, 12> { using ZPZ = aerobus::zpz<241>; using type =
04024
                         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<10>, ZPZV<109>, ZPZV<168>, ZPZV<22>,
                         ZPZV<197>, ZPZV<17>, ZPZV<7»; }; // NOLINT</pre>
04025
                                       template<> struct ConwayPolynomial<241, 13> { using ZPZ = aerobus::zpz<241>; using type =
                        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<234»; }; // NOLINT template<> struct ConwayPolynomial<241, 17> { using ZPZ = aerobus::zpz<241>; using type =
04026
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<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
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<11>, ZPZV<234»; }; //</pre>
                       NOLINT
                                      template<> struct ConwayPolynomial<251, 1> { using ZPZ = aerobus::zpz<251>; using type =
                      POLYV<ZPZV<1>, ZPZV<245»; }; // NOLINT
                                    template<> struct ConwayPolynomial<251, 2> { using ZPZ = aerobus::zpz<251>; using type =
                     POLYV<ZPZV<1>, ZPZV<242>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<251, 3> { using ZPZ = aerobus::zpz<251>; using type =
04030
                      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<245»; }; // NOLINT template<> struct ConwayPolynomial<251, 4> { using ZPZ = aerobus::zpz<251>; using type =
04031
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<200>, ZPZV<6»; }; // NOLINT
04032
                                    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 template<> struct ConwayPolynomial<251, 6> { using ZPZ = aerobus::zpz<251>; using type =
04033
                      template<> struct ConwayFolynomial<221, 6> (using ZFZ - aerobus::ZpZZ231>; using type POLYVZPZVX1>, ZPZV<1>, ZPZV<1>, ZPZV<24>, ZPZV<151>, ZPZV<19>, ZPZV<6>; ; // NOLINT 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<0>, ZPZV<8>, ZPZV<84, ZPZV<245»; };
                                     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»; }; //
                                    template<> struct ConwayPolynomial<251, 9> { using ZPZ = aerobus::zpz<251>; using type =
04036
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<187>, ZPZV<106>, ZPZV<245»;
                       }; // NOLINT
                                    template<> struct ConwayPolynomial<251, 10> { using ZPZ = aerobus::zpz<251>; using type =
04037
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<118>, ZPZV<110>, ZPZV<45>, ZPZV<34>,
                       ZPZV<149>, ZPZV<6»; }; // NOLINT</pre>
                                     template<> struct ConwayPolynomial<251, 11> { using ZPZ = aerobus::zpz<251>; using type =
04038
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<26>, ZPZV<245»; };</pre>
                                                                                                                  // NOLINT
                                      template<> struct ConwayPolynomial<251, 12> { using ZPZ = aerobus::zpz<251>; using type
04039
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<192>, ZPZV<53>, ZPZV<20>, ZPZV<20>, ZPZV<15>,
                       \mbox{ZPZV}\mbox{<201>, ZPZV}\mbox{<232>, ZPZV}\mbox{<6}\mbox{*; }; \mbox{$//$ NOLINT$}
                                       template<> struct ConwayPolynomial<251, 13> { using ZPZ = aerobus::zpz<251>; using type =
04040
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                       template<> struct ConwayPolynomial<251,
                                                                                                                                                                                              17> { using ZPZ = aerobus::zpz<251>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                                                                                                                                                                              // NOLINT
                       template<> struct ConwayPolynomial<251, 19> { using ZPZ = aerobus::zpz<251>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
04042
                        ZPZV<0>, ZPZV<245*; }; //</pre>
                       NOLINT
04043
                                      template<> struct ConwayPolynomial<257, 1> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<254»; }; // NOLINT
                                     template<> struct ConwayPolynomial<257, 2> { using ZPZ = aerobus::zpz<257>; using type =
04044
                      POLYV<ZPZV<1>, ZPZV<251>, ZPZV<3»; }; // NOLINT
                                      template<> struct ConwayPolynomial<257, 3> { using ZPZ = aerobus::zpz<257>; using type =
04045
                     POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 4> { using ZPZ = aerobus::zpz<257>; using type =
                       \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<16>, \ \verb"ZPZV<187>, \ \verb"ZPZV<3"; \ \verb"}; \ \ // \ \verb"NOLINT" 
04047
                                     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
                                      template<> struct ConwayPolynomial<257, 6> { using ZPZ = aerobus::zpz<257>; using type =
04048
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<62>, ZPZV<18>, ZPZV<18>, ZPZV<138>, ZPZV<3»; }; // NOLINT
                                    template<> struct ConwayPolynomial<257, 7> { using ZPZ = aerobus::zpz<257>; using type =
04049
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<25+, ZPZV<254»; }; // NOLINT template<> struct ConwayPolynomial<257, 8> { using ZPZ = aerobus::zpz<257>; using type =
04050
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<179>, ZPZV<140>, ZPZV<162>, ZPZV<3»; }; //
                       NOLINT
                                      template<> struct ConwayPolynomial<257, 9> { using ZPZ = aerobus::zpz<257>; using type
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<201>, ZPZV<50>, ZPZV<254»;
04052
                                    template<> struct ConwayPolynomial<257, 10> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<12>, ZPZV<225>, ZPZV<180>, ZPZV<20>,
                       ZPZV<3»: }: // NOLINT
                                     template<> struct ConwayPolynomial<257, 11> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<40>, ZPZV<254»; }; // NOLINT</pre>
04054
                                   template<> struct ConwayPolynomial<257, 12> { using ZPZ = aerobus::zpz<257>; using type =
                      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<23>, ZPZV<225>, ZPZV<215>, ZPZV<2173>, ZPZV<249>, ZPZV<148>, ZPZV<20>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<257, 13> { using ZPZ = aerobus::zpz<257>; using type =
04055
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                       ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<254»; };</pre>
                                                                                                                                                                                       // NOLINT
04056
                                     template<> struct ConwayPolynomial<257, 17> { using ZPZ = aerobus::zpz<257>; using type =
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                      ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<54>; ); // NOLINT
template<> struct ConwayPolynomial<257, 19> { using ZPZ = aerobus::zpz<257>; using type
04057
                       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                        ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<254»; }; //</pre>
                       NOLINT
04058
                                     template<> struct ConwayPolynomial<263, 1> { using ZPZ = aerobus::zpz<263>; using type =
                      POLYV<ZPZV<1>, ZPZV<258»; }; // NOLINT
                                     template<> struct ConwayPolynomial<263, 2> { using ZPZ = aerobus::zpz<263>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<261>, ZPZV<5»; };
                        template<> struct ConwayPolynomial<263, 3> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 4> { using ZPZ = aerobus::zpz<263>; using type =
04061
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<171>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<263, 5> { using ZPZ = aerobus::zpz<263>; using type =
04062
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<258»; }; // NOLINT
                        template<> struct ConwayPolynomial<263, 6> { using ZPZ = aerobus::zpz<263>; using type =
04063
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<22>, ZPZV<250>, ZPZV<25>, ZPZV<55»; }; // NOLINT template<> struct ConwayPolynomial<263, 7> { using ZPZ = aerobus::zpz<263>; using type =
04064
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<258»; }; // NOLINT template<> struct ConwayPolynomial<263, 8> { using ZPZ = aerobus::zpz<263>; using type =
04065
              POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<227>, ZPZV<170>, ZPZV<7>, ZPZV<5»; };
04066
                      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<6>, ZPZV<6>, ZPZV<261>, ZPZV<29>, ZPZV<258»;
              }; // NOLINT
              template<> struct ConwayPolynomial<263, 10> { using ZPZ = aerobus::zpz<263>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<245>, ZPZV<231>, ZPZV<198>, ZPZV<145>,
04067
              ZPZV<119>, ZPZV<5»; };</pre>
                                                                        // NOLINT
                        template<> struct ConwayPolynomial<263, 11> { using ZPZ = aerobus::zpz<263>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<2>, ZPZV<258»; }; // NOLINT</pre>
              template<> struct ConwayPolynomial<263, 12> { using ZPZ = aerobus::zpz<263>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<174>, ZPZV<162>, ZPZV<252>, ZPZV<47>, ZPZV<45>, ZPZV<180>, ZPZV<5»; }; // NOLINT
04069
                      template<> struct ConwayPolynomial<269, 1> { using ZPZ = aerobus::zpz<269>; using type =
04070
              POLYV<ZPZV<1>, ZPZV<267»; }; // NOLINT
04071
                        template<> struct ConwayPolynomial<269, 2> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<268>, ZPZV<2»; }; // NOLINT
                       template<> struct ConwayPolynomial<269, 3> { using ZPZ = aerobus::zpz<269>; using type =
04072
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<267»; }; // NOLINT template<> struct ConwayPolynomial<269, 4> { using ZPZ = aerobus::zpz<269>; using type =
04073
              04074
                        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
             template<> struct ConwayPolynomial<269, 6> { using ZPZ = aerobus::zpz<269>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<10>, ZPZV<206>, ZPZV<2»; }; // NOLINT
04075
04076
                        template<> struct ConwayPolynomial<269, 7> { using ZPZ = aerobus::zpz<269>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 - , 
04077
                      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<23x; }; //
              NOLINT
04078
                       template<> struct ConwayPolynomial<269, 9> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<214>, ZPZV<267>, ZPZV<267»;
              }; // NOLINT
04079
                       template<> struct ConwayPolynomial<269, 10> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<264>, ZPZV<243>, ZPZV<186>, ZPZV<61>, ZPZV<10>, ZPZV<20>; }; // NOLINT
                        template<> struct ConwayPolynomial<269, 11> { using ZPZ = aerobus::zpz<269>; using type =
04080
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
              ZPZV<20>, ZPZV<267»; }; // NOLINT</pre>
04081
                        template<> struct ConwayPolynomial<269, 12> { using ZPZ = aerobus::zpz<269>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2), ZPZV<165>, ZPZV<165>, ZPZV<165>, ZPZV<63>, ZPZV<215>, ZPZV<132>, ZPZV<180>, ZPZV<150>, ZPZV<2; }; // NOLINT

template<> struct ConwayPolynomial<271, 1> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<265»; }; // NOLINT
04082
                      template<> struct ConwayPolynomial<271, 2> { using ZPZ = aerobus::zpz<271>; using type =
04083
              POLYV<ZPZV<1>, ZPZV<269>, ZPZV<6»; }; // NOLINT
04084
                        template<> struct ConwayPolynomial<271, 3> { using ZPZ = aerobus::zpz<271>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<265»; ); // NOLINT
template<> struct ConwayPolynomial<271, 4> { using ZPZ = aerobus::zpz<271>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<205>, ZPZV<6»; }; // NOLINT
04085
                        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
04087
                       template<> struct ConwayPolynomial<271, 6> { using ZPZ = aerobus::zpz<271>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<207>, ZPZV<207>, ZPZV<81>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<271, 7> { using ZPZ = aerobus::zpz<271>; using type :
04088
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<265»; }; // NOLINT
                        template<> struct ConwayPolynomial<271, 8> { using ZPZ = aerobus::zpz<271>; using type
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<199>, ZPZV<114>, ZPZV<69>, ZPZV<6»; };
              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<266>, ZPZV<186>, ZPZV<266>,
ZPZV<266>, ZPZV<266>, ZPZV<266>, ZPZV<266>, ZPZV<266>, ZPZV<266>, ZPZV<266>, ZPZV<266>, ZPZV<266</pre>
04090
              }; // NOLINT
    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»; };</pre>
                                                                     // NOLINT
                        template<> struct ConwayPolynomial<271, 11> { using ZPZ = aerobus::zpz<271>; using type =
              POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10, ZPZV<265»; }; // NOLINT 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
04094
                      template<> struct ConwayPolynomial<277, 1> { using ZPZ = aerobus::zpz<277>; using type =
             POLYV<ZPZV<1>, ZPZV<272»; }; // NOLINT
                      template<> struct ConwayPolynomial<277, 2> { using ZPZ = aerobus::zpz<277>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<274>, ZPZV<5»; };
                          template<> struct ConwayPolynomial<277, 3> { using ZPZ = aerobus::zpz<277>; using type =
               POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<272»; }; // NOLINT template<> struct ConwayPolynomial<277, 4> { using ZPZ = aerobus::zpz<277>; using type =
04097
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<222>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<277, 5> { using ZPZ = aerobus::zpz<277>; using type =
04098
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<272»; }; // NOLINT
04099
                           template<> struct ConwayPolynomial<277, 6> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<33>, ZPZV<9>, ZPZV<118>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<277, 7> { using ZPZ = aerobus::zpz<277>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<27>; // NOLINT
template<> struct ConwayPolynomial<277, 8> { using ZPZ = aerobus::zpz<277>; using type =
04100
04101
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<159>, ZPZV<176>, ZPZV<176>, ZPZV<5»; }; //
04102
                        template<> struct ConwayPolynomial<277, 9> { using ZPZ = aerobus::zpz<277>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<177>, ZPZV<110>, ZPZV<272»;
                }; // NOLINT
               template<> struct ConwayPolynomial<277, 10> { using ZPZ = aerobus::zpz<277>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<206>, ZPZV<253>, ZPZV<237>, ZPZV<241>,
04103
                ZPZV<260>, ZPZV<5»; };</pre>
                                                                                // NOLINT
                          template<> struct ConwayPolynomial<277, 11> { using ZPZ = aerobus::zpz<277>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                ZPZV<5>, ZPZV<272»; }; // NOLINT</pre>
               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<40>, ZPZV<180>, ZPZV<115>, ZPZV<202>, ZPZV<5»; }; // NOLINT
04105
                         template<> struct ConwayPolynomial<281, 1> { using ZPZ = aerobus::zpz<281>; using type =
04106
               POLYV<ZPZV<1>, ZPZV<278»; }; // NOLINT
04107
                          template<> struct ConwayPolynomial<281, 2> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<280>, ZPZV<3»; }; // NOLINT
                         template<> struct ConwayPolynomial<281, 3> { using ZPZ = aerobus::zpz<281>; using type =
04108
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<278»; }; // NOLINT template<> struct ConwayPolynomial<281, 4> { using ZPZ = aerobus::zpz<281>; using type =
04109
               04110
                          template<> struct ConwayPolynomial<281, 5> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<278»; }; // NOLINT
               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
04111
04112
                           template<> struct ConwayPolynomial<281,
                                                                                                                                    7> { using ZPZ = aerobus::zpz<281>; using type
               POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<278»; }; //
04113
                         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
04114
                         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<6>, ZPZV<6>, ZPZV<6>, ZPZV<70>, 
                }; // NOLINT
04115
                          template<> struct ConwayPolynomial<281, 10> { using ZPZ = aerobus::zpz<281>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<258>, ZPZV<145>, ZPZV<13>, ZPZV<138>,
                ZPZV<191>, ZPZV<3»: }: // NOLINT
                          template<> struct ConwayPolynomial<281, 11> { using ZPZ = aerobus::zpz<281>; using type =
04116
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>,
                ZPZV<36>, ZPZV<278»; }; // NOLINT</pre>
04117
                           template<> struct ConwayPolynomial<281, 12> { using ZPZ = aerobus::zpz<281>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<202>, ZPZV<68>, ZPZV<68>, ZPZV<116>, ZPZV<58>, ZPZV<28>, ZPZV<191>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<283, 1> { using ZPZ = aerobus::zpz<283>; using type =
04118
               POLYV<ZPZV<1>, ZPZV<280»; }; // NOLINT
                         template<> struct ConwayPolynomial<283, 2> { using ZPZ = aerobus::zpz<283>; using type =
04119
               POLYV<ZPZV<1>, ZPZV<282>, ZPZV<3»; }; // NOLINT
04120
                           template<> struct ConwayPolynomial<283, 3> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<280»; ); // NOLINT
template<> struct ConwayPolynomial<283, 4> { using ZPZ = aerobus::zpz<283>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<238>, ZPZV<3»; }; // NOLINT
04121
                           template<> struct ConwayPolynomial<283, 5> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<280»; }; // NOLINT
04123
                         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
04124
                         template<> struct ConwayPolynomial<283, 7> { using ZPZ = aerobus::zpz<283>; using type =
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<88, ZPZV<280»; }; // NOLINT
                           template<> struct ConwayPolynomial<283, 8> { using ZPZ = aerobus::zpz<283>; using type
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<179>, ZPZV<32>, ZPZV<232>, ZPZV<23»; }; //
               template<> struct ConwayPolynomial<283, 9> { using ZPZ = aerobus::zpz<283>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<65>, ZPZV<280»;</pre>
04126
               }; // NOLINT
  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»; };</pre>
                                                                             // NOLINT
                           template<> struct ConwayPolynomial<283, 11> { using ZPZ = aerobus::zpz<283>; using type =
                POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
                ZPZV<4>, ZPZV<280»; }; // NOLINT</pre>
                           template<> struct ConwayPolynomial<283, 12> { using ZPZ = aerobus::zpz<283>; using type
               POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<8>, ZPZV<96>, ZPZV<229>, ZPZV<49>, ZPZV<14>, ZPZV<56>, ZPZV<3»; }; // NOLINT
04130
                         \texttt{template<>} \texttt{struct ConwayPolynomial} < 293, \ 1> \ \{ \texttt{using ZPZ = aerobus::zpz} < 293>; \texttt{using type = aerobus::zpz
              POLYV<ZPZV<1>, ZPZV<291»; }; // NOLINT
04131
                         template<> struct ConwayPolynomial<293, 2> { using ZPZ = aerobus::zpz<293>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<292>, ZPZV<2»; };
                      template<> struct ConwayPolynomial<293, 3> { using ZPZ = aerobus::zpz<293>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT template<> struct ConwayPolynomial<293, 4> { using ZPZ = aerobus::zpz<293>; using type =
04133
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<166>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<293, 5> { using ZPZ = aerobus::zpz<293>; using type =
04134
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<291»; }; // NOLINT
                      template<> struct ConwayPolynomial<293, 6> { using ZPZ = aerobus::zpz<293>; using type =
04135
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<210>, ZPZV<260>, ZPZV<20>; }; // NOLINT template<> struct ConwayPolynomial<293, 7> { using ZPZ = aerobus::zpz<293>; using type =
04136
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>; ZPZV<0>, ZPZV<0>; ZPZV<0 ; ZPZV<0
04137
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<29>, ZPZV<175>, ZPZV<195>, ZPZV<239>, ZPZV<2»; }; //
04138
                     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<0>, ZPZV<208>, ZPZV<190>, ZPZV<291»;
             }; // NOLINT
             template<> struct ConwayPolynomial<293, 10> { using ZPZ = aerobus::zpz<293>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<28>, ZPZV<46>, ZPZV<184>, ZPZV<24>,
04139
             ZPZV<2»; }; // NOLINT</pre>
                      template<> struct ConwayPolynomial<293, 11> { using ZPZ = aerobus::zpz<293>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
             ZPZV<3>, ZPZV<291»; }; // NOLINT</pre>
             template<> struct ConwayPolynomial<293, 12> { using ZPZ = aerobus::zpz<293>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<210>, ZPZV<12>, ZPZV<167>, ZPZV<144>, ZPZV<157>, ZPZV<22»; }; // NOLINT
04141
                     template<> struct ConwayPolynomial<307, 1> { using ZPZ = aerobus::zpz<307>; using type =
04142
             POLYV<ZPZV<1>, ZPZV<302»; }; // NOLINT
04143
                      template<> struct ConwayPolynomial<307, 2> { using ZPZ = aerobus::zpz<307>; using type =
             POLYV<ZPZV<1>, ZPZV<306>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<307, 3> { using ZPZ = aerobus::zpz<307>; using type =
04144
             POLYY<ZPZY<1>, ZPZY<0>, ZPZY<7>, ZPZY<302»; }; // NOLINT template<> struct ConwayPolynomial<307, 4> { using ZPZ = aerobus::zpz<307>; using type =
04145
             04146
                      template<> struct ConwayPolynomial<307, 5> { using ZPZ = aerobus::zpz<307>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<302»; }; // NOLINT
             template<> struct ConwayPolynomial</pr>
307, 6> { using ZPZ = aerobus::zpz<307>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<213>, ZPZV<61>, ZPZV<61>, ZPZV<5»; }; // NOLINT</pre>
04147
04148
                      template<> struct ConwayPolynomial<307, 7> { using ZPZ = aerobus::zpz<307>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6 , ZPZV<6
04149
                     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
04150
                     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<165>, ZPZV<70>, ZPZV<302»;
             }; // NOLINT
04151
                     template<> struct ConwayPolynomial<311, 1> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<ZPZV<1>, ZPZV<294»; }; // NOLINT
                      template<> struct ConwayPolynomial<311, 2> { using ZPZ = aerobus::zpz<311>; using type =
04152
             POLYV<ZPZV<1>, ZPZV<310>, ZPZV<17»; }; // NOLINT
04153
                      template<> struct ConwayPolynomial<311, 3> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<294»; }; // NOLINT
04154
                      template<> struct ConwayPolynomial<311, 4> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<163>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<311, 5> { using ZPZ = aerobus::zpz<311>; using type =
04155
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<294»; }; // NOLINT template<> struct ConwayPolynomial<311, 6> { using ZPZ = aerobus::zpz<311>; using type =
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<27>, ZPZV<167>, ZPZV<152>, ZPZV<17»; }; // NOLINT
                      template<> struct ConwayPolynomial<311, 7> { using ZPZ = aerobus::zpz<311>; using type =
04157
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<294»; }; // NOLT template<> struct ConwayPolynomial<311, 8> { using ZPZ = aerobus::zpz<311>; using type =
04158
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<19>, ZPZV<162>, ZPZV<118>, ZPZV<2>, ZPZV<17»; }; //
             NOLINT
                      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<284, ZPZV<294»;
             }; // NOLINT
04160
                      template<> struct ConwayPolynomial<313, 1> { using ZPZ = aerobus::zpz<313>; using type =
             POLYV<ZPZV<1>, ZPZV<303»; }; // NOLINT
                      template<> struct ConwayPolynomial<313, 2> { using ZPZ = aerobus::zpz<313>; using type =
04161
             POLYV<ZPZV<1>, ZPZV<310>, ZPZV<10»; }; // NOLINT
                      template<> struct ConwayPolynomial<313, 3> { using ZPZ = aerobus::zpz<313>; using type =
04162
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<303»; }; // NOLINT template<> struct ConwayPolynomial<313, 4> { using ZPZ = aerobus::zpz<313>; using type =
04163
             POLYY<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<29>, ZPZV<29; ; // NOLINT
template<> struct ConwayPolynomial<313, 5> { using ZPZ = aerobus::zpz<313>; using type =
04164
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<303»; }; // NOLINT
                      template<> struct ConwayPolynomial<313, 6> { using ZPZ = aerobus::zpz<313>; using type =
04165
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<196>, ZPZV<213>, ZPZV<253>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<313, 7> { using ZPZ = aerobus::zpz<313>; using type
04166
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<303»; }; // NOLINT
                     template<> struct ConwayPolynomial<313, 8> { using ZPZ = aerobus::zpz<313>; using type =
04167
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<306>, ZPZV<99>, ZPZV<106>, ZPZV<10»; }; //
04168
                     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<8>, ZPZV<267>, ZPZV<300>, ZPZV<303»;
             }; // NOLINT
04169
                    template<> struct ConwayPolynomial<317, 1> { using ZPZ = aerobus::zpz<317>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<315»; };
                      template<> struct ConwayPolynomial<317, 2> { using ZPZ = aerobus::zpz<317>; using type =
04170
            POLYV<ZPZV<1>, ZPZV<313>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<317, 3> { using ZPZ = aerobus::zpz<317>; using type =
04171
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<315»; }; // NOLINT template<> struct ConwayPolynomial<317, 4> { using ZPZ = aerobus::zpz<317>; using type =
04172
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<178>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<317, 5> { using ZPZ = aerobus::zpz<317>; using type =
04173
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<315»; // NOLINT
            template<> struct ConwayPolynomial<317, 6> { using ZPZ = aerobus::zpz<317>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<195>, ZPZV<156>, ZPZV<4>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<317, 7> { using ZPZ = aerobus::zpz<317>; using type =
04174
04175
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<315»; }; // NoLII template<> struct ConwayPolynomial<317, 8> { using ZPZ = aerobus::zpz<317>; using type = aerobus::zpz<317>;
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<207>, ZPZV<85>, ZPZV<31>, ZPZV<2»; };
                     template<> struct ConwayPolynomial<317, 9> { using ZPZ = aerobus::zpz<317>; using type =
04177
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<284>, ZPZV<296>, ZPZV<315»;
             }; // NOLINT
                      template<> struct ConwayPolynomial<331, 1> { using ZPZ = aerobus::zpz<331>; using type =
             POLYV<ZPZV<1>, ZPZV<328»; }; // NOLINT
                      template<> struct ConwayPolynomial<331, 2> { using ZPZ = aerobus::zpz<331>; using type =
04179
             POLYV<ZPZV<1>, ZPZV<326>, ZPZV<3%; }; // NOLINT template<> struct ConwayPolynomial<331, 3> { using ZPZ = aerobus::zpz<331>; using type =
04180
            POLYV<ZPZV<1>, ZPZV<2>, ZPZV<1>, ZPZV<328»; }; // NOLINT template<> struct ConwayPolynomial<331, 4> { using ZPZ = aerobus::zpz<331>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<290>, ZPZV<3»; }; // NOLINT
04182
                      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
                     template<> struct ConwayPolynomial<331, 6> { using ZPZ = aerobus::zpz<331>; using type =
04183
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<205>, ZPZV<159>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<331, 7> { using ZPZ = aerobus::zpz<331>; using type
04184
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
                    template<> struct ConwayPolynomial<331, 8> { using ZPZ = aerobus::zpz<331>; using type =
04185
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<24>>, ZPZV<308>, ZPZV<78>, ZPZV<3»; };
             NOLINT
             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»;
04186
             }; // NOLINT
                      template<> struct ConwayPolynomial<337, 1> { using ZPZ = aerobus::zpz<337>; using type =
04187
            POLYV<ZPZV<1>, ZPZV<327»; }; // NOLINT
                      template<> struct ConwayPolynomial<337, 2> { using ZPZ = aerobus::zpz<337>; using type =
04188
             POLYV<ZPZV<1>, ZPZV<332>, ZPZV<10»; }; // NOLINT template<> struct ConwayPolynomial<337, 3> { using ZPZ = aerobus::zpz<337>; using type =
04189
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT
04190
                      template<> struct ConwayPolynomial<337, 4> { using ZPZ = aerobus::zpz<337>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<25>, ZPZV<224>, ZPZV<10»; }; // NOLINT
                     template<> struct ConwayPolynomial<337, 5> { using ZPZ = aerobus::zpz<337>; using type =
04191
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<327»; }; // NOLINT template<> struct ConwayPolynomial<337, 6> { using ZPZ = aerobus::zpz<337>; using type =
04192
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<216>, ZPZV<127>, ZPZV<109>, ZPZV<10»; }; // NOLINT
                      template<> struct ConwayPolynomial<337, 7> { using ZPZ = aerobus::zpz<337>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5-, ZPZV<5
04194
                     template<> struct ConwayPolynomial<337, 8> { using ZPZ = aerobus::zpz<337>; using type =
             POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<331>, ZPZV<246>, ZPZV<251>, ZPZV<10»; }; //
             NOLINT
                      template<> struct ConwayPolynomial<337, 9> { using ZPZ = aerobus::zpz<337>; using type =
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<148>, ZPZV<98>, ZPZV<327»;
             }; // NOLINT
04196
                      template<> struct ConwayPolynomial<347, 1> { using ZPZ = aerobus::zpz<347>; using type =
             POLYV<ZPZV<1>, ZPZV<345»; }; // NOLINT
                     template<> struct ConwayPolynomial<347, 2> { using ZPZ = aerobus::zpz<347>; using type =
04197
            POLYV<ZPZV<1>, ZPZV<343>, ZPZV<2»; }; // NOLINT
                      template<> struct ConwayPolynomial<347, 3> { using ZPZ = aerobus::zpz<347>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 4> { using ZPZ = aerobus::zpz<347>; using type =
04199
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<13>, ZPZV<295>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<347, 5> { using ZPZ = aerobus::zpz<347>; using type =
04200
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<345»; }; // NOLINT
04201
                      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
04202
                    template<> struct ConwayPolynomial<347, 7> { using ZPZ = aerobus::zpz<347>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<345»; }; // NOLINT template<> struct ConwayPolynomial<347, 8> { using ZPZ = aerobus::zpz<347>; using type =
04203
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<187>, ZPZV<213>, ZPZV<117>, ZPZV<2»; }; //
04204
                     template<> struct ConwayPolynomial<347, 9> { using ZPZ = aerobus::zpz<347>; using type
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<235>, ZPZV<252>, ZPZV<252>, ZPZV<345»;
             }; // NOLINT
04205
                      template<> struct ConwayPolynomial<349, 1> { using ZPZ = aerobus::zpz<349>; using type =
             POLYV<ZPZV<1>, ZPZV<347»; }; // NOLINT
                      template<> struct ConwayPolynomial<349, 2> { using ZPZ = aerobus::zpz<349>; using type =
            POLYV<ZPZV<1>, ZPZV<348>, ZPZV<2»; }; // NOLINT
04207
                     template<> struct ConwayPolynomial<349, 3> { using ZPZ = aerobus::zpz<349>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<347»; }; // NOLINT template<> struct ConwayPolynomial<349, 4> { using ZPZ = aerobus::zpz<349>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<279>, ZPZV<2»; }; // NOLINT
04208
```

```
04209
               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
              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<2x; }; // NOLINT template<> struct ConwayPolynomial<349, 7> { using ZPZ = aerobus::zpz<349>; using type
04211
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<347»; };
                                                                                                                                         // NOLINT
               template<> struct ConwayPolynomial<349, 8> { using ZPZ = aerobus::zpz<349>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<328>, ZPZV<268>, ZPZV<268>, ZPZV<268; }; //
        template<> struct ConwayPolynomial<349, 9> { using ZPZ = aerobus::zpz<349>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<36>, ZPZV<290>, ZPZV<130>, ZPZV<347»;
04213
         }; // NOLINT
04214
               template<> struct ConwayPolynomial<353, 1> { using ZPZ = aerobus::zpz<353>; using type =
         POLYV<ZPZV<1>, ZPZV<350»; }; // NOLINT
04215
               template<> struct ConwayPolynomial<353, 2> { using ZPZ = aerobus::zpz<353>; using type =
        POLYV<ZPZV<1>, ZPZV<348>, ZPZV<3w; }; // NOLINT template<> struct ConwayPolynomial<353, 3> { using ZPZ = aerobus::zpz<353>; using type =
04216
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<350»; }; // NOLINT template<> struct ConwayPolynomial<353, 4> { using ZPZ = aerobus::zpz<353>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<199>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<353, 5> { using ZPZ = aerobus::zpz<353>; using type =
04218
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<350»; }; // NOLINT
04219
              template<> struct ConwayPolynomial<353, 6> { using ZPZ = aerobus::zpz<353>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<26>, ZPZV<295>, ZPZV<29s, ; ; // NOLINT template<> struct ConwayPolynomial<353, 7> { using ZPZ = aerobus::zpz<353>; using type
04220
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<350»; }; // NOLINT
04221
               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
04222
              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
               template<> struct ConwayPolynomial<359, 1> { using ZPZ = aerobus::zpz<359>; using type =
04223
         POLYV<ZPZV<1>, ZPZV<352»; }; // NOLINT
               template<> struct ConwayPolynomial<359, 2> { using ZPZ = aerobus::zpz<359>; using type =
04224
        POLYV<ZPZV<1>, ZPZV<358>, ZPZV<7»; }; // NOLINT
               template<> struct ConwayPolynomial<359, 3> { using ZPZ = aerobus::zpz<359>; using type =
04225
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<352»; }; // NOLINT
04226
               template<> struct ConwayPolynomial<359, 4> { using ZPZ = aerobus::zpz<359>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<22>, ZPZY<229>, ZPZY<7»; }; // NOLINT template<> struct ConwayPolynomial<359, 5> { using ZPZ = aerobus::zpz<359>; using type =
04227
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<3>, ZPZV<352»; }; // NOLINT template<> struct ConwayPolynomial<359, 6> { using ZPZ = aerobus::zpz<359>; using type = POLYV<ZPZV<1>, ZPZV<3>, ZPZV<3>, ZPZV<327>, ZPZV<327>, ZPZV<7»; }; // NOLINT
04228
               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<0>, ZPZV<0>, ZPZV<1>, ZPZV<352»; };
04230
              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
               template<> struct ConwayPolynomial<359, 9> { using ZPZ = aerobus::zpz<359>; using type =
04231
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<356>, ZPZV<165>, ZPZV<352»;
         }; // NOLINT
04232
               template<> struct ConwayPolynomial<367, 1> { using ZPZ = aerobus::zpz<367>; using type =
        POLYV<ZPZV<1>, ZPZV<361»; }; // NOLINT
              template<> struct ConwayPolynomial<367, 2> { using ZPZ = aerobus::zpz<367>; using type =
04233
        POLYV<ZPZV<1>, ZPZV<366>, ZPZV<6»; }; // NOLINT
               template<> struct ConwayPolynomial<367, 3> { using ZPZ = aerobus::zpz<367>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<361»; }; // NOLINT
               template<> struct ConwayPolynomial<367, 4> { using ZPZ = aerobus::zpz<367>; using type =
04235
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<295>, ZPZV<6»; }; // NOLINT
        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
04236
04237
               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<324>, ZPZV<6»; }; // NOLINT
04238
              template<> struct ConwayPolynomial<367, 7> { using ZPZ = aerobus::zpz<367>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361»; }; // NOLINT template<> struct ConwayPolynomial<367, 8> { using ZPZ = aerobus::zpz<367>; using type =
04239
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<335>, ZPZV<282>, ZPZV<50>, ZPZV<6»; }; //
         NOLINT
04240
               template<> struct ConwayPolynomial<367, 9> { using ZPZ = aerobus::zpz<367>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<15>, ZPZV<213>, ZPZV<268>, ZPZV<361»;
         }; // NOLINT
04241
               template<> struct ConwayPolynomial<373, 1> { using ZPZ = aerobus::zpz<373>; using type =
        POLYV<ZPZV<1>, ZPZV<371»; }; // NOLINT
               template<> struct ConwayPolynomial<373, 2> { using ZPZ = aerobus::zpz<373>; using type =
04242
         POLYV<ZPZV<1>, ZPZV<369>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<373, 3> { using ZPZ = aerobus::zpz<373>; using type =
04243
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<371»; }; // NOLINT template<> struct ConwayPolynomial<373, 4> { using ZPZ = aerobus::zpz<373>; using type =
04244
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<304>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<373, 5> { using ZPZ = aerobus::zpz<373>; using type =
04245
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<371»; }; // NOLINT
               template<> struct ConwayPolynomial<373, 6> { using ZPZ = aerobus::zpz<373>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<126>, ZPZV<83>, ZPZV<108>, ZPZV<2»; }; // NOLINT
04247
              template<> struct ConwayPolynomial<373, 7> { using ZPZ = aerobus::zpz<373>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
04248
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<203>, ZPZV<219>, ZPZV<66>, ZPZV<2»; }; //
04249
           template<> struct ConwayPolynomial<373, 9> { using ZPZ = aerobus::zpz<373>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<238>, ZPZV<2370>, ZPZV<371»;
       }; // NOLINT
04250
           template<> struct ConwayPolynomial<379, 1> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<377»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 2> { using ZPZ = aerobus::zpz<379>; using type =
04251
       POLYV<ZPZV<1>, ZPZV<374>, ZPZV<2»; }; // NOLINT
04252
           template<> struct ConwayPolynomial<379, 3> { using ZPZ = aerobus::zpz<379>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<377»; }; // NOLINT template<> struct ConwayPolynomial<379, 4> { using ZPZ = aerobus::zpz<379>; using type =
04253
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<327>, ZPZV<2»; };
                                                                        // NOLINT
            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
04255
           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<2½; }; // NOLINT template<> struct ConwayPolynomial<379, 7> { using ZPZ = aerobus::zpz<379>; using type
04256
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<377»; }; // NOLINT
           template<> struct ConwayPolynomial<379, 8> { using ZPZ = aerobus::zpz<379>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<210>, ZPZV<194>, ZPZV<173>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<379, 9> { using ZPZ = aerobus::zpz<379>; using type =
04258
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<362>, ZPZV<369>, ZPZV<377»;
       }; // NOLINT
            template<> struct ConwayPolynomial<383, 1> { using ZPZ = aerobus::zpz<383>; using type =
      POLYV<ZPZV<1>, ZPZV<378»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 2> { using ZPZ = aerobus::zpz<383>; using type =
04260
      POLYV<ZPZV<1>, ZPZV<382>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<383, 3> { using ZPZ = aerobus::zpz<383>; using type =
04261
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 4> { using ZPZ = aerobus::zpz<383>; using type =
04262
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<309>, ZPZV<5»; }; // NOLINT
          template<> struct ConwayPolynomial<383, 5> { using ZPZ = aerobus::zpz<383>; using type =
04263
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<378»; }; // NOLINT template<> struct ConwayPolynomial<383, 6> { using ZPZ = aerobus::zpz<383>; using type =
04264
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<69>, ZPZV<85, ZPZV<58>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<383, 7> { using ZPZ = aerobus::zpz<383>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<378»; };
           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
           template<> struct ConwayPolynomial<383, 9> { using ZPZ = aerobus::zpz<383>; using type =
04267
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<137>, ZPZV<76>, ZPZV<378»;
04268
           template<> struct ConwayPolynomial<389, 1> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<387»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 2> { using ZPZ = aerobus::zpz<389>; using type =
04269
      POLYV<ZPZV<1>, ZPZV<379>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 3> { using ZPZ = aerobus::zpz<389>; using type =
04270
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<387»; }; // NOLINT template<> struct ConwayPolynomial<389, 4> { using ZPZ = aerobus::zpz<389>; using type =
      04272
           template<> struct ConwayPolynomial<389, 5> { using ZPZ = aerobus::zpz<389>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<387»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 6> { using ZPZ = aerobus::zpz<389>; using type =
04273
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<339>, ZPZV<255>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<389, 7> { using ZPZ = aerobus::zpz<389>; using type
04274
      POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<2>, ZPZY<2>, ZPZY<28, ZPZY<287; // NOLI template<> struct ConwayPolynomial<389, 8> { using ZPZ = aerobus::zpz<389>; using type =
04275
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<351>, ZPZV<19>, ZPZV<290>, ZPZV<2»; }; //
       NOLINT
04276
           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
04277
           template<> struct ConwayPolynomial<397, 1> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<392»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 2> { using ZPZ = aerobus::zpz<397>; using type =
04278
      POLYV<ZPZV<1>, ZPZV<392>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 3> { using ZPZ = aerobus::zpz<397>; using type =
04279
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<392»; }; // NOLINT template<> struct ConwayPolynomial<397, 4> { using ZPZ = aerobus::zpz<397>; using type =
04280
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<363>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<397, 5> { using ZPZ = aerobus::zpz<397>; using type =
04281
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<392»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 6> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<382>, ZPZV<274>, ZPZV<287>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<397, 7> { using ZPZ = aerobus::zpz<397>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<392»; }; // NOLINT template<> struct ConwayPolynomial<397, 8> { using ZPZ = aerobus::zpz<397>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<25>, ZPZV<203>, ZPZV<5»; }; //
04284
      template<> struct ConwayPolynomial<397, 9> { using ZPZ = aerobus::zpz<397>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<166>, ZPZV<166>, ZPZV<252>, ZPZV<392»;
       }; // NOLINT
04286
           template<> struct ConwayPolynomial<401, 1> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<398»; }; // NOLINT
```

```
04287
            template<> struct ConwayPolynomial<401, 2> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<396>, ZPZV<3»; }; // NOLINT
04288
           template<> struct ConwayPolynomial<401, 3> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<398»; }; // NOLINT
template<> struct ConwayPolynomial<401, 4> { using ZPZ = aerobus::zpz<401>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<372>, ZPZV<3»; }; // NOLINT
04289
            template<> struct ConwayPolynomial 401, 5> { using ZPZ = aerobus::zpz<401>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<398»; }; // NOLINT
           template<> struct ConwayPolynomial<401, 6> { using ZPZ = aerobus::zpz<401>; using type =
04291
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<115>, ZPZV<81>, ZPZV<51>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<401, 7> { using ZPZ = aerobus::zpz<401>; using type =
04292
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<398»; }; // NOLINT
           template<> struct ConwayPolynomial<401, 8> { using ZPZ = aerobus::zpz<401>; using type =
04293
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<380>, ZPZV<113>, ZPZV<164>, ZPZV<3»; }; //
       NOLINT
04294
           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
            template<> struct ConwayPolynomial<409, 1> { using ZPZ = aerobus::zpz<409>; using type =
       POLYV<ZPZV<1>, ZPZV<388»; }; // NOLINT
            template<> struct ConwayPolynomial<409, 2> { using ZPZ = aerobus::zpz<409>; using type =
       POLYV<ZPZV<1>, ZPZV<404>, ZPZV<21»; }; // NOLINT
           template<> struct ConwayPolynomial<409, 3> { using ZPZ = aerobus::zpz<409>; using type =
04297
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<388»; }; // NOLINT template<> struct ConwayPolynomial<409, 4> { using ZPZ = aerobus::zpz<409>; using type =
04298
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<407>, ZPZV<21»; }; // NOLINT
04299
           template<> struct ConwayPolynomial<409, 5> { using ZPZ = aerobus::zpz<409>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<388»; }; // NOLINT
04300
            template<> struct ConwayPolynomial<409, 6> { using ZPZ = aerobus::zpz<409>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<372>, ZPZV<535>, ZPZV<364>, ZPZV<21»; }; // NOLINT template<> struct ConwayPolynomial<409, 7> { using ZPZ = aerobus::zpz<409>; using type
04301
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<388»; };
           template<> struct ConwayPolynomial<409, 8> { using ZPZ = aerobus::zpz<409>; using type =
04302
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<56>, ZPZV<69>, ZPZV<396>, ZPZV<31»; }; //
       NOLINT
04303
           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<0>, ZPZV<8>, ZPZV<818>, ZPZV<318>, ZPZV<211>, ZPZV<388»;
04304
            template<> struct ConwayPolynomial<419, 1> { using ZPZ = aerobus::zpz<419>; using type =
       POLYV<ZPZV<1>, ZPZV<417»; }; // NOLINT
04305
           template<> struct ConwayPolynomial<419, 2> { using ZPZ = aerobus::zpz<419>; using type =
      POLYV<ZPZV<1>, ZPZV<418>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<419, 3> { using ZPZ = aerobus::zpz<419>; using type =
04306
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<417»; }; // NOLINT template<> struct ConwayPolynomial<419, 4> { using ZPZ = aerobus::zpz<419>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<373>, ZPZV<2»; }; // NOLINT
04308
           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
           template<> struct ConwayPolynomial<419, 6> { using ZPZ = aerobus::zpz<419>; using type =
04309
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<411>, ZPZV<33>, ZPZV<257>, ZPZV<2»; }; // NOLINT
04310
           template<> struct ConwayPolynomial<419,
                                                         7> { using ZPZ = aerobus::zpz<419>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<417»; }; // NOLIN template<> struct ConwayPolynomial<419, 8> { using ZPZ = aerobus::zpz<419>; using type =
04311
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<234>, ZPZV<388>, ZPZV<151>, ZPZV<2»; }; //
       NOLINT
       template<> struct ConwayPolynomial<419, 9> { using ZPZ = aerobus::zpz<419>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<386>, ZPZV<417»;</pre>
04312
       }; // NOLINT
            template<> struct ConwayPolynomial<421, 1> { using ZPZ = aerobus::zpz<421>; using type =
04313
       POLYV<ZPZV<1>, ZPZV<419»; }; // NOLINT
           template<> struct ConwayPolynomial<421, 2> { using ZPZ = aerobus::zpz<421>; using type =
04314
       POLYV<ZPZV<1>, ZPZV<417>, ZPZV<2»; }; // NOLINT
04315
            template<> struct ConwayPolynomial<421, 3> { using ZPZ = aerobus::zpz<421>; using type =
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 4> { using ZPZ = aerobus::zpz<421>; using type =
04316
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<257>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<421, 5> { using ZPZ = aerobus::zpz<421>; using type =
04317
       POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZV<0>, ZPZV<15>, ZPZV<419»; }; // NOLINT template<> struct ConwayPolynomial<421, 6> { using ZPZ = aerobus::zpz<421>; using type =
04318
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<111>, ZPZV<342>, ZPZV<41>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<421, 7> { using ZPZ = aerobus::zpz<421>; using type
04319
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<21>, ZPZV<21>, ZPZV<419»; };
04320
           template<> struct ConwayPolynomial<421, 8> { using ZPZ = aerobus::zpz<421>; using type =
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<389>, ZPZV<32>, ZPZV<77>, ZPZV<2»; };
       NOLINT
           template<> struct ConwayPolynomial<421, 9> { using ZPZ = aerobus::zpz<421>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<18>, ZPZV<394>, ZPZV<145>, ZPZV<419»;
       }; // NOLINT
04322
           template<> struct ConwayPolynomial<431, 1> { using ZPZ = aerobus::zpz<431>; using type =
       POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<431, 2> { using ZPZ = aerobus::zpz<431>; using type =
04323
       POLYV<ZPZV<1>, ZPZV<430>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<431, 3> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<424»; };
                                                              // NOLINT
           template<> struct ConwayPolynomial<431, 4> { using ZPZ = aerobus::zpz<431>; using type =
04325
      POLYV<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<7»; }; // NOLINT 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»; };
           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
04328
           template<> struct ConwayPolynomial<431, 7> { using ZPZ = aerobus::zpz<431>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT template<> struct ConwayPolynomial<431, 8> { using ZPZ = aerobus::zpz<431>; using type =
04329
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<243>, ZPZV<286>, ZPZV<115>, ZPZV<7»; }; //
04330
           template<> struct ConwayPolynomial<431, 9> { using ZPZ = aerobus::zpz<431>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<71>, ZPZV<329>, ZPZV<424%;
       }; // NOLINT
04331
           template<> struct ConwayPolynomial<433, 1> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<428»; }; // NOLINT
            template<> struct ConwayPolynomial<433, 2> { using ZPZ = aerobus::zpz<433>; using type =
       POLYV<ZPZV<1>, ZPZV<432>, ZPZV<5»; }; // NOLINT
04333
           template<> struct ConwayPolynomial<433, 3> { using ZPZ = aerobus::zpz<433>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<428»; }; // NOLINT template<> struct ConwayPolynomial<433, 4> { using ZPZ = aerobus::zpz<433>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<402>, ZPZV<5»; }; // NOLINT
04334
           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
04336
           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<55; }; // NOLINT template<> struct ConwayPolynomial<433, 7> { using ZPZ = aerobus::zpz<433>; using type
04337
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<428»; };
           template<> struct ConwayPolynomial<433, 8> { using ZPZ = aerobus::zpz<433>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<347>, ZPZV<32>, ZPZV<39>, ZPZV<5»; };
       NOLINT
      template<> struct ConwayPolynomial<433, 9> { using ZPZ = aerobus::zpz<433>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<27>, ZPZV<232>, ZPZV<45>, ZPZV<428»;</pre>
04339
       }; // NOLINT
04340
            template<> struct ConwayPolynomial<439, 1> { using ZPZ = aerobus::zpz<439>; using type =
       POLYV<ZPZV<1>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 2> { using ZPZ = aerobus::zpz<439>; using type =
04341
      POLYV<ZPZV<1>, ZPZV<436>, ZPZV<15»; }; // NOLINT template<> struct ConwayPolynomial<439, 3> { using ZPZ = aerobus::zpz<439>; using type =
04342
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<424»; }; // NOLINT
           template<> struct ConwayPolynomial<439, 4> { using ZPZ = aerobus::zpz<439>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<323>, ZPZV<15»; };
                                                                          // NOLINT
           template<> struct ConwayPolynomial<439, 5> { using ZPZ = aerobus::zpz<439>; using type =
04344
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<424»; }; // NOLINT
      template<> struct ConwayPolynomial439, 6> { using ZPZ = aerobus::zpz<39>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<324>, ZPZV<190>, ZPZV<15»; }; // NOLINT
04345
04346
           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
04347
           template<> struct ConwayPolynomial<439, 8> { using ZPZ = aerobus::zpz<439>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<35>, ZPZV<296>, ZPZV<266>, ZPZV<15»; }; //
       NOLINT
           template<> struct ConwayPolynomial<439, 9> { using ZPZ = aerobus::zpz<439>; using type =
04348
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<342>, ZPZV<254>, ZPZV<424»;
       }; // NOLINT
           template<> struct ConwayPolynomial<443, 1> { using ZPZ = aerobus::zpz<443>; using type =
04349
      POLYY<ZPZY<1>, ZPZY<441>; }; // NOLINT template<> struct ConwayPolynomial<443, 2> { using ZPZ = aerobus::zpz<443>; using type =
04350
      POLYV<ZPZV<1>, ZPZV<437>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 3> { using ZPZ = aerobus::zpz<443>; using type =
04351
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT
           template<> struct ConwayPolynomial<443, 4> { using ZPZ = aerobus::zpz<443>; using type =
04352
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<383>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<443, 5> { using ZPZ = aerobus::zpz<443>; using type =
04353
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<441»; }; // NOLINT template<> struct ConwayPolynomial<443, 6> { using ZPZ = aerobus::zpz<443>; using type =
04354
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<298>, ZPZV<218>, ZPZV<41>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<443, 7> { using ZPZ = aerobus::zpz<443>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<441»; };
04356
           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
           template<> struct ConwayPolynomial<443, 9> { using ZPZ = aerobus::zpz<443>; using type =
04357
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<125>, ZPZV<109, ZPZV<441»;
       }; // NOLINT
04358
           template<> struct ConwayPolynomial<449, 1> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<446»; }; // NOLINT
           template<> struct ConwayPolynomial<449, 2> { using ZPZ = aerobus::zpz<449>; using type =
04359
      POLYV<ZPZV<1>, ZPZV<444>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<449, 3> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 4> { using ZPZ = aerobus::zpz<449>; using type =
04361
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<249>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<449, 5> { using ZPZ = aerobus::zpz<449>; using type =
04362
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<446»; }; // NOLINT
04363
           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»; }; // NOLINI
04364
           template<> struct ConwayPolynomial<449, 7> { using ZPZ = aerobus::zpz<449>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<446»; }; // NOLINT template<> struct ConwayPolynomial<449, 8> { using ZPZ = aerobus::zpz<449>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<361>, ZPZV<348>, ZPZV<124>, ZPZV<3»; }; //
04365
```

```
NOLINT
           template<> struct ConwayPolynomial<449, 9> { using ZPZ = aerobus::zpz<449>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<26>, ZPZV<3>, ZPZV<446»; };
            // NOLINT
04367
                   template<> struct ConwayPolynomial<457, 1> { using ZPZ = aerobus::zpz<457>; using type =
           POLYV<ZPZV<1>, ZPZV<444»; }; // NOLINT
                   template<> struct ConwayPolynomial<457, 2> { using ZPZ = aerobus::zpz<457>; using type =
           POLYV<ZPZV<1>, ZPZV<454>, ZPZV<13»; }; // NOLINT
                  template<> struct ConwayPolynomial<457, 3> { using ZPZ = aerobus::zpz<457>; using type =
04369
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<444»; }; // NOLINT template<> struct ConwayPolynomial<457, 4> { using ZPZ = aerobus::zpz<457>; using type =
04370
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<407>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<457, 5> { using ZPZ = aerobus::zpz<457>; using type =
04371
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<44>, ZPZV<444»; }; // NOLINT
04372
                  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 template<> struct ConwayPolynomial<457, 7> { using ZPZ = aerobus::zpz<457>; using type :
04373
           POLYVCZPZVC1>, ZPZV<0>, ZPZV<0
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<365>, ZPZV<296>, ZPZV<412>, ZPZV<13»; }; //
04375
                   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<0>, ZPZV<5>, ZPZV<354>, ZPZV<844, ZPZV<444*;
           }; // NOLINT
04376
                   template<> struct ConwayPolynomial<461, 1> { using ZPZ = aerobus::zpz<461>; using type =
           POLYV<ZPZV<1>, ZPZV<459»; }; // NOLINT
04377
                  template<> struct ConwayPolynomial<461, 2> { using ZPZ = aerobus::zpz<461>; using type =
           POLYV<ZPZV<1>, ZPZV<460>, ZPZV<2»; }; // NOLINT
04378
                   template<> struct ConwayPolynomial<461, 3> { using ZPZ = aerobus::zpz<461>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<459»; }; // NOLINT
template<> struct ConwayPolynomial<461, 4> { using ZPZ = aerobus::zpz<461>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<393>, ZPZV<2»; }; // NOLINT
04379
                   template<> struct ConwayPolynomial<461, 5> { using ZPZ = aerobus::zpz<461>; using type =
04380
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<459»; }; // NOLINT
04381
                   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 template<> struct ConwayPolynomial<461, 7> { using ZPZ = aerobus::zpz<461>; using type :
04382
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<459»; }; //
                   template<> struct ConwayPolynomial<461, 8> { using ZPZ = aerobus::zpz<461>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<388>, ZPZV<449>, ZPZV<321>, ZPZV<2»; }; //
           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<210>, ZPZV<210>, ZPZV<216>, ZPZV<25>,
04384
           }; // NOLINT
                   template<> struct ConwayPolynomial<463, 1> { using ZPZ = aerobus::zpz<463>; using type =
           POLYV<ZPZV<1>, ZPZV<460»; }; // NOLINT
04386
                  template<> struct ConwayPolynomial<463, 2> { using ZPZ = aerobus::zpz<463>; using type =
           POLYV<ZPZV<1>, ZPZV<461>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<463, 3> { using ZPZ = aerobus::zpz<463>; using type =
04387
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<10>, ZPZY<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 4> { using ZPZ = aerobus::zpz<463>; using type =
04388
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<17>, ZPZV<262>, ZPZV<3»; }; // NOLINT
04389
                   template<> struct ConwayPolynomial<463, 5> { using ZPZ = aerobus::zpz<463>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<460»; }; // NOLINT template<> struct ConwayPolynomial<463, 6> { using ZPZ = aerobus::zpz<463>; using type =
04390
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<462>, ZPZV<51>, ZPZV<110>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<463, 7> { using ZPZ = aerobus::zpz<463>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<460»; };
                  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
           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<43>, ZPZV<433>, ZPZV<433>, ZPZV<227>, ZPZV<460»;
04393
           }; // NOLINT
                  template<> struct ConwayPolynomial<467, 1> { using ZPZ = aerobus::zpz<467>; using type =
04394
           POLYV<ZPZV<1>, ZPZV<465»; }; // NOLINT
04395
                   template<> struct ConwayPolynomial<467, 2> { using ZPZ = aerobus::zpz<467>; using type =
           POLYV<ZPZV<1>, ZPZV<463>, ZPZV<2»; }; // NOLINT
                   template<> struct ConwayPolynomial<467, 3> { using ZPZ = aerobus::zpz<467>; using type =
04396
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 4> { using ZPZ = aerobus::zpz<467>; using type =
04397
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<353>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<467, 5> { using ZPZ = aerobus::zpz<467>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<465»; }; // NOLINT
04398
                   template<> struct ConwayPolynomial<467, 6> { using ZPZ = aerobus::zpz<467>; using type =
04399
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<123>, ZPZV<62>, ZPZV<237>, ZPZV<2»; }; // NOLINI
                   template<> struct ConwayPolynomial<467, 7> { using ZPZ = aerobus::zpz<467>; using type
04400
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<465»; }; // NOLINT template<> struct ConwayPolynomial<467, 8> { using ZPZ = aerobus::zpz<467>; using type =
04401
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<318>, ZPZV<413>, ZPZV<289>, ZPZV<2»; }; //
           NOLINT
04402
                   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<44<sup>7</sup>>, ZPZV<465»;
           }; // NOLINT
04403
                   \texttt{template<> struct ConwayPolynomial<479, 1> \{ using ZPZ = aerobus:: zpz<479>; using type = 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 200 + 20
           POLYV<ZPZV<1>, ZPZV<466»; }; // NOLINT
                  template<> struct ConwayPolynomial<479, 2> { using ZPZ = aerobus::zpz<479>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<474>, ZPZV<13»; };
                                                                       // NOLINT
               template<> struct ConwayPolynomial<479, 3> { using ZPZ = aerobus::zpz<479>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<466»; }; // NOLINT template<> struct ConwayPolynomial<479, 4> { using ZPZ = aerobus::zpz<479>; using type =
04406
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<386>, ZPZV<13»; }; // NOLINT template<> struct ConwayPolynomial<479, 5> { using ZPZ = aerobus::zpz<479>; using type =
04407
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<466»; }; // NOLINT
               template<> struct ConwayPolynomial<479, 6> { using ZPZ = aerobus::zpz<479>; using type
04408
          \texttt{POLYV} < \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 0>, \ \texttt{ZPZV} < 1>, \ \texttt{ZPZV} < 243>, \ \texttt{ZPZV} < 287>, \ \texttt{ZPZV} < 334>, \ \texttt{ZPZV} < 13 \\ \text{*}; \ \ // \ \ \texttt{NOLINT} 
04409
              template<> struct ConwayPolynomial<479, 7> { using ZPZ = aerobus::zpz<479>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<46»; }; // NOLINT template<> struct ConwayPolynomial<479, 8> { using ZPZ = aerobus::zpz<479>; using type =
04410
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<247>, ZPZV<440>, ZPZV<17>, ZPZV<13»; }; //
04411
              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
04412
               template<> struct ConwayPolynomial<487, 1> { using ZPZ = aerobus::zpz<487>; using type =
         POLYV<ZPZV<1>, ZPZV<484»; }; // NOLINT
               template<> struct ConwayPolynomial<487, 2> { using ZPZ = aerobus::zpz<487>; using type =
         POLYV<ZPZV<1>, ZPZV<485>, ZPZV<3»; }; // NOLINT
04414
               template<> struct ConwayPolynomial<487, 3> { using ZPZ = aerobus::zpz<487>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<484»; }; // NOLINT
template<> struct ConwayPolynomial<487, 4> { using ZPZ = aerobus::zpz<487>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<483>, ZPZV<3»; }; // NOLINT
04415
               template<> struct ConwayPolynomial<487, 5> { using ZPZ = aerobus::zpz<487>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<484»; }; // NOLINT
04417
               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
04418
              template<> struct ConwayPolynomial<487, 7> { using ZPZ = aerobus::zpz<487>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<0>, ZPZV<18484»; }; // NOLINT template<> struct ConwayPolynomial<487, 8> { using ZPZ = aerobus::zpz<487>; using type =
04419
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<283>, ZPZV<249>, ZPZV<137>, ZPZV<3»; }; //
         template<> struct ConwayPolynomial<487, 9> { using ZPZ = aerobus::zpz<487>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<271>, ZPZV<4447>, ZPZV<484»;</pre>
04420
         }; // NOLINT
               template<> struct ConwayPolynomial<491, 1> { using ZPZ = aerobus::zpz<491>; using type =
         POLYV<ZPZV<1>, ZPZV<489»; }; // NOLINT
               template<> struct ConwayPolynomial<491, 2> { using ZPZ = aerobus::zpz<491>; using type =
        POLYV<ZPZV<1>, ZPZV<487>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<491, 3> { using ZPZ = aerobus::zpz<491>; using type =
04423
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<489»; }; // NOLINT template<> struct ConwayPolynomial<491, 4> { using ZPZ = aerobus::zpz<491>; using type =
04424
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<360>, ZPZV<2»; }; // NOLINT
04425
               template<> struct ConwayPolynomial<491, 5> { using ZPZ = aerobus::zpz<491>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; }; // NOLINT
04426
              template<> struct ConwayPolynomial<491, 6> { using ZPZ = aerobus::zpz<491>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<489»; };
               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»; }; //
04429
              template<> struct ConwayPolynomial<491, 9> { using ZPZ = aerobus::zpz<491>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<453>, ZPZV<489»;
04430
              template<> struct ConwayPolynomial<499, 1> { using ZPZ = aerobus::zpz<499>; using type =
         POLYV<ZPZV<1>, ZPZV<492»; }; // NOLINT
               template<> struct ConwayPolynomial<499, 2> { using ZPZ = aerobus::zpz<499>; using type =
04431
         POLYV<ZPZV<1>, ZPZV<493>, ZPZV<7»; }; // NOLINT
              template<> struct ConwayPolynomial<499, 3> { using ZPZ = aerobus::zpz<499>; using type =
04432
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<492»; }; // NOLINT template<> struct ConwayPolynomial<499, 4> { using ZPZ = aerobus::zpz<499>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<495>, ZPZV<7»; }; // NOLINT
04434
              template<> struct ConwayPolynomial<499, 5> { using ZPZ = aerobus::zpz<499>; using type =
          \verb"POLYV<ZPZV<1>, \verb"ZPZV<0>, \verb"ZPZV<0>, \verb"ZPZV<1>, \verb"ZPZV<492"; \verb"}; $ // \verb"NOLINT" | NOLINT" 
        template<> struct ConwayPolynomial<499, 6> { using ZPZ = aerobus::zpz<499>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<407>, ZPZV<191>, ZPZV<78>, ZPZV<7»; }; // NOLINT</pre>
04435
04436
               template<> struct ConwayPolynomial<499, 7> { using ZPZ = aerobus::zpz<499>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<492»; }; // NOLIN template<> struct ConwayPolynomial<499, 8> { using ZPZ = aerobus::zpz<499>; using type =
04437
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<288>, ZPZV<309>, ZPZV<200>, ZPZV<7»; }; //
         NOLINT
              template<> struct ConwayPolynomial<499, 9> { using ZPZ = aerobus::zpz<499>; using type =
04438
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<491>, ZPZV<491>, ZPZV<492»;
         }; // NOLINT
04439
               template<> struct ConwayPolynomial<503, 1> { using ZPZ = aerobus::zpz<503>; using type =
         POLYV<ZPZV<1>, ZPZV<498»; }; // NOLINT
               template<> struct ConwayPolynomial<503, 2> { using ZPZ = aerobus::zpz<503>; using type =
04440
         POLYV<ZPZV<1>, ZPZV<498>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<503, 3> { using ZPZ = aerobus::zpz<503>; using type =
         POLYY<ZPZY<1>, ZPZV<0>, ZPZV<2>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 4> { using ZPZ = aerobus::zpz<503>; using type =
04442
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<325>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 5> { using ZPZ = aerobus::zpz<503>; using type =
04443
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<498»; }; // NOLINT
```

```
template<> struct ConwayPolynomial<503, 6> { using ZPZ = aerobus::zpz<503>; using type =
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<380>, ZPZV<292>, ZPZV<255>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<503, 7> { using ZPZ = aerobus::zpz<503>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<498»; }; // NOLINT template<> struct ConwayPolynomial<503, 8> { using ZPZ = aerobus::zpz<503>; using type =
04446
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<441>, ZPZV<203>, ZPZV<316>, ZPZV<5»; }; //
            NOLINT
04447
                    template<> struct ConwayPolynomial<503, 9> { using ZPZ = aerobus::zpz<503>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<158>, ZPZV<337>, ZPZV<498»;
            }; // NOLINT
04448
                   template<> struct ConwayPolynomial<509, 1> { using ZPZ = aerobus::zpz<509>; using type =
           POLYV<ZPZV<1>, ZPZV<507»; }; // NOLINT
                   template<> struct ConwayPolynomial<509, 2> { using ZPZ = aerobus::zpz<509>; using type =
04449
           POLYV<ZPZV<1>, ZPZV<508>, ZPZV<2»; }; // NOLINT
04450
                   template<> struct ConwayPolynomial<509, 3> { using ZPZ = aerobus::zpz<509>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
template<> struct ConwayPolynomial<509, 4> { using ZPZ = aerobus::zpz<509>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<408>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<509, 5> { using ZPZ = aerobus::zpz<509>; using type =
04451
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<507»; }; // NOLINT
                    template<> struct ConwayPolynomial<509, 6> { using ZPZ = aerobus::zpz<509>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<350>, ZPZV<232>, ZPZV<41>, ZPZV<2»; }; // NOLINT
                   template<> struct ConwayPolynomial<509, 7> { using ZPZ = aerobus::zpz<509>; using type =
04454
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>), ZPZV<0>, ZPZV<0>), 
04455
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<420>, ZPZV<473>, ZPZV<382>, ZPZV<382>; }; //
           template<> struct ConwayPolynomial<509, 9> { using ZPZ = aerobus::zpz<509>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<314>, ZPZV<28>, ZPZV<507»;</pre>
04456
            }; // NOLINT
                    template<> struct ConwayPolynomial<521, 1> { using ZPZ = aerobus::zpz<521>; using type =
04457
           POLYV<ZPZV<1>, ZPZV<518»; }; // NOLINT
                   template<> struct ConwayPolynomial<521, 2> { using ZPZ = aerobus::zpz<521>; using type =
04458
            POLYV<ZPZV<1>, ZPZV<515>, ZPZV<3»; }; // NOLINT
04459
                    template<> struct ConwayPolynomial<521, 3> { using ZPZ = aerobus::zpz<521>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<518»; }; // NOLINT
template<> struct ConwayPolynomial<521, 4> { using ZPZ = aerobus::zpz<521>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<509>, ZPZV<3»; }; // NOLINT
04460
04461
                    template<> struct ConwayPolynomial<521, 5> { using ZPZ = aerobus::zpz<521>; using type =
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<518»; }; // NOLINT
           template<> struct ConwayPolynomial<521, 6> { using ZPZ = aerobus::zpz<521>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<315>, ZPZV<153>, ZPZV<280>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<521, 7> { using ZPZ = aerobus::zpz<521>; using type =
04462
04463
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0 , ZPZV<0
                   template<> struct ConwayPolynomial<521, 8> { using ZPZ = aerobus::zpz<521>; using type
            POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<462>, ZPZV<407>, ZPZV<312>, ZPZV<33»; }; //
           template<> struct ConwayPolynomial<521, 9> { using ZPZ = aerobus::zpz<521>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<181>, ZPZV<483>, ZPZV<518»;</pre>
04465
            }; // NOLINT
04466
                    template<> struct ConwayPolynomial<523, 1> { using ZPZ = aerobus::zpz<523>; using type =
            POLYV<ZPZV<1>, ZPZV<521»; }; // NOLINT
04467
                    template<> struct ConwayPolynomial<523, 2> { using ZPZ = aerobus::zpz<523>; using type =
           POLYV<ZPZV<1>, ZPZV<522>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<523, 3> { using ZPZ = aerobus::zpz<523>; using type =
04468
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<5>, ZPZY<521»; }; // NOLINT template<> struct ConwayPolynomial<523, 4> { using ZPZ = aerobus::zpz<523>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<382>, ZPZV<2»; }; // NOLINT
                    template<> struct ConwayPolynomial<523, 5> { using ZPZ = aerobus::zpz<523>; using type =
04470
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<521»; }; // NOLINT
                   template<> struct ConwayPolynomial<523, 6> { using ZPZ = aerobus::zpz<523>; using type =
04471
           POLYVCZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<475>, ZPZV<371>, ZPZV<371>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<523, 7> { using ZPZ = aerobus::zpz<523>; using type
            POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<521»; }; //
04473
                   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
04474
                   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<1>, ZPZV<14>, ZPZV<342>, ZPZV<145>, ZPZV<521»;
            }; // NOLINT
                   template<> struct ConwayPolynomial<541, 1> { using ZPZ = aerobus::zpz<541>; using type =
04475
           POLYV<ZPZV<1>, ZPZV<539»; }; // NOLINT
                   template<> struct ConwayPolynomial<541, 2> { using ZPZ = aerobus::zpz<541>; using type =
04476
           POLYV<ZPZV<1>, ZPZV<537>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<541, 3> { using ZPZ = aerobus::zpz<541>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<539»; }; // NOLINT
04477
                   template<> struct ConwayPolynomial<541, 4> { using ZPZ = aerobus::zpz<541>; using type =
04478
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<333>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<541, 5> { using ZPZ = aerobus::zpz<541>; using type =
04479
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<539»; }; // NOLINT
           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
04480
                    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»; };
04482
                  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<113>, ZPZV<22; }; //
            NOLTNT
```

```
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
04484
           template<> struct ConwayPolynomial<547, 1> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<545»; }; // NOLINT
           template<> struct ConwayPolynomial<547, 2> { using ZPZ = aerobus::zpz<547>; using type =
04485
      POLYV<ZPZV<1>, ZPZV<543>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<547, 3> { using ZPZ = aerobus::zpz<547>; using type =
04486
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 4> { using ZPZ = aerobus::zpz<547>; using type =
04487
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<334>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<547, 5> { using ZPZ = aerobus::zpz<547>; using type =
04488
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<545»; }; // NOLINT
           template<> struct ConwayPolynomial<547, 6> { using ZPZ = aerobus::zpz<547>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<334>, ZPZV<153>, ZPZV<423>, ZPZV<2»; }; // NOLINI
04490
           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<0>, ZPZV<1>, ZPZV<545»; }; // NOLINT template<> struct ConwayPolynomial<547, 8> { using ZPZ = aerobus::zpz<547>; using type =
04491
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<368>, ZPZV<20>, ZPZV<180>, ZPZV<2»; }; //
      template<> struct ConwayPolynomial<547, 9> { using ZPZ = aerobus::zpz<547>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<238>, ZPZV<263>, ZPZV<245»;
04492
       }; // NOLINT
           template<> struct ConwayPolynomial<557, 1> { using ZPZ = aerobus::zpz<557>; using type =
04493
      POLYV<ZPZV<1>, ZPZV<555»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 2> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<553>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<557, 3> { using ZPZ = aerobus::zpz<557>; using type =
04495
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<555»; }; // NOLINT template<> struct ConwayPolynomial<557, 4> { using ZPZ = aerobus::zpz<557>; using type =
04496
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<430>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<557, 5> { using ZPZ = aerobus::zpz<557>; using type =
04497
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<555»; }; // NOLINT
04498
          template<> struct ConwayPolynomial<557, 6> { using ZPZ = aerobus::zpz<557>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<20>, ZPZV<192>, ZPZV<253>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<557, 7> { using ZPZ = aerobus::zpz<557>; using type =
04499
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<555»; }; // NOLINT 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»; };
04501
          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
04502
           template<> struct ConwayPolynomial<563, 1> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<561»; }; // NOLINT
04503
           template<> struct ConwayPolynomial<563, 2> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<559>, ZPZV<2»; }; // NOLINT
04504
           template<> struct ConwayPolynomial<563, 3> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<561»; }; // NOLINT
           template<> struct ConwayPolynomial<563, 4> { using ZPZ = aerobus::zpz<563>; using type =
04505
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<20>, ZPZV<399>, ZPZV<2»; };
                                                                         // NOLINT
           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
04507
           template<> struct ConwayPolynomial<563, 6> { using ZPZ = aerobus::zpz<563>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<12>, ZPZV<303, ZPZV<246>, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<563, 7> { using ZPZ = aerobus::zpz<563>; using type
04508
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<561»; }; //
          template<> struct ConwayPolynomial<563, 8> { using ZPZ = aerobus::zpz<563>; using type =
04509
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<176>, ZPZV<509>, ZPZV<2»; }; //
       NOLINT
           template<> struct ConwayPolynomial<563, 9> { using ZPZ = aerobus::zpz<563>; using type =
04510
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<15>, ZPZV<15>, ZPZV<19>, ZPZV<561»; };
       // NOLINT
           template<> struct ConwayPolynomial<569, 1> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<566»; }; // NOLINT
04512
           template<> struct ConwayPolynomial<569, 2> { using ZPZ = aerobus::zpz<569>; using type =
      POLYV<ZPZV<1>, ZPZV<568>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial5569, 3> { using ZPZ = aerobus::zpz<569>; using type =
04513
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 4> { using ZPZ = aerobus::zpz<569>; using type =
04514
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<381>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<569, 5> { using ZPZ = aerobus::zpz<569>; using type =
04515
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<566»; }; // NOLINT template<> struct ConwayPolynomial<569, 6> { using ZPZ = aerobus::zpz<569>; using type =
04516
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<50>, ZPZV<480>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<569, 7> { using ZPZ = aerobus::zpz<569>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<56%; };
04518
           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
           template<> struct ConwayPolynomial<569, 9> { using ZPZ = aerobus::zpz<569>; using type =
04519
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<478>, ZPZV<566>, ZPZV<566»;
       }; // NOLINT
04520
           template<> struct ConwayPolynomial<571, 1> { using ZPZ = aerobus::zpz<571>; using type =
      POLYV<ZPZV<1>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 2> { using ZPZ = aerobus::zpz<571>; using type =
04521
       POLYV<ZPZV<1>, ZPZV<570>, ZPZV<3»; }; // NOLINT
```

```
04522
               template<> struct ConwayPolynomial<571, 3> { using ZPZ = aerobus::zpz<571>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 4> { using ZPZ = aerobus::zpz<571>; using type =
04523
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<402>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<571, 5> { using ZPZ = aerobus::zpz<571>; using type =
04524
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<568»; }; // NOLINT template<> struct ConwayPolynomial<571, 6> { using ZPZ = aerobus::zpz<571>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<221>, ZPZV<295>, ZPZV<33>, ZPZV<3»; }; // NOLINI
              template<> struct ConwayPolynomial<571, 7> { using ZPZ = aerobus::zpz<571>; using type =
04526
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<568»; }; // NOLINT
04527
              template<> struct ConwayPolynomial<571, 8> { using ZPZ = aerobus::zpz<571>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<363>, ZPZV<119>, ZPZV<371>, ZPZV<37: }; //
               template<> struct ConwayPolynomial<571, 9> { using ZPZ = aerobus::zpz<571>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<34>, ZPZV<545>, ZPZV<179>, ZPZV<568»;
         }; // NOLINT
              template<> struct ConwayPolynomial<577, 1> { using ZPZ = aerobus::zpz<577>; using type =
04529
        POLYV<ZPZV<1>, ZPZV<572»; }; // NOLINT
               template<> struct ConwayPolynomial<577, 2> { using ZPZ = aerobus::zpz<577>; using type =
        POLYV<ZPZV<1>, ZPZV<572>, ZPZV<5»; }; // NOLINT
               template<> struct ConwayPolynomial<577, 3> { using ZPZ = aerobus::zpz<577>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<572»; }; // NOLINT template<> struct ConwayPolynomial<577, 4> { using ZPZ = aerobus::zpz<577>; using type =
04532
        POLYY<ZPZV<1>, ZPZV<1>, ZPZV<494, ZPZV<494, ZPZV<494, ZPZV<5; }; // NOLINT
template<> struct ConwayPolynomial<577, 5> { using ZPZ = aerobus::zpz<577>; using type =
04533
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<572»; }; // NOLINT
              template<> struct ConwayPolynomial<577, 6> { using ZPZ = aerobus::zpz<577>; using type =
04534
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<450>, ZPZV<25>, ZPZV<283>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<577, 7> { using ZPZ = aerobus::zpz<577>; using type :
04535
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8>, ZPZV<572»; }; // NOLINT
              template<> struct ConwayPolynomial<577, 8> { using ZPZ = aerobus::zpz<577>; using type =
04536
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<450>, ZPZV<545>, ZPZV<321>, ZPZV<32; //
              template<> struct ConwayPolynomial<577, 9> { using ZPZ = aerobus::zpz<577>; using type =
04537
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<576>, ZPZV<449>, ZPZV<572»;
         }; // NOLINT
        template<> struct ConwayPolynomial<587, 1> { using ZPZ = aerobus::zpz<587>; using type =
POLYV<ZPZV<1>, ZPZV<585»; }; // NOLINT</pre>
04538
               template<> struct ConwayPolynomial<587, 2> { using ZPZ = aerobus::zpz<587>; using type =
         POLYV<ZPZV<1>, ZPZV<583>, ZPZV<2»; }; // NOLINT
04540
              template<> struct ConwayPolynomial<587, 3> { using ZPZ = aerobus::zpz<587>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<585»; }; // NOLINT
template<> struct ConwayPolynomial<587, 4> { using ZPZ = aerobus::zpz<587>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<16>, ZPZV<444>, ZPZV<2»; }; // NOLINT
04541
               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
04543
              template<> struct ConwayPolynomial<587, 6> { using ZPZ = aerobus::zpz<587>; using type =
         \texttt{POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<204>, ZPZV<121>, ZPZV<226>, ZPZV<2<math>\texttt{s}; \texttt{}; \texttt{}// \texttt{NOLINT} 
              template<> struct ConwayPolynomial<587, 7> { using ZPZ = aerobus::zpz<587>; using type =
04544
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<585»; }; // NOLINT
               template<> struct ConwayPolynomial<587, 8> { using ZPZ = aerobus::zpz<587>; using type
04545
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<49>, ZPZV<492>, ZPZV<444>, ZPZV<91>, ZPZV<91; };
         NOLINT
        \label{eq:convayPolynomial} $$ template<> struct ConwayPolynomial<587, 9> { using ZPZ = aerobus::zpz<587>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<333>, ZPZV<55>, ZPZV<585»; ZPZV<0>, ZPZV<
04546
         }; // NOLINT
04547
               template<> struct ConwayPolynomial<593, 1> { using ZPZ = aerobus::zpz<593>; using type =
        POLYV<ZPZV<1>, ZPZV<590»; }; // NOLINT
               template<> struct ConwayPolynomial<593, 2> { using ZPZ = aerobus::zpz<593>; using type =
        POLYV<ZPZV<1>, ZPZV<592>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<593, 3> { using ZPZ = aerobus::zpz<593>; using type =
04549
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<590»; }; // NOLINT template<> struct ConwayPolynomial<593, 4> { using ZPZ = aerobus::zpz<593>; using type =
04550
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<419>, ZPZV<3»; }; // NOLINT
04551
              template<> struct ConwayPolynomial<593, 5> { using ZPZ = aerobus::zpz<593>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<590»; }; // NOLINT
04552
              template<> struct ConwayPolynomial<593, 6> { using ZPZ = aerobus::zpz<593>; using type =
        POLYY<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<345>, ZPZV<478>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<593, 7> { using ZPZ = aerobus::zpz<593>; using type
04553
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<590»; }; //
04554
              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<49; }; //
         NOLINT
              template<> struct ConwayPolynomial<593, 9> { using ZPZ = aerobus::zpz<593>; using type =
04555
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<223>, ZPZV<523>, ZPZV<590»;
         }; // NOLINT
04556
               template<> struct ConwayPolynomial<599, 1> { using ZPZ = aerobus::zpz<599>; using type =
         POLYV<ZPZV<1>, ZPZV<592»; }; // NOLINT
              template<> struct ConwayPolynomial<599, 2> { using ZPZ = aerobus::zpz<599>; using type =
04557
        POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
              template<> struct ConwayPolynomial<599, 3> { using ZPZ = aerobus::zpz<599>; using type =
04558
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<592»; }; // NOLINT template<> struct ConwayPolynomial<599, 4> { using ZPZ = aerobus::zpz<599>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<419>, ZPZV<7»; }; // NOLINT
04560
             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
04561
              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»; };
           template<> struct ConwayPolynomial<599, 7> { using ZPZ = aerobus::zpz<599>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<592»; }; // NOLINT
04563
           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»; }; //
       NOT.TNT
04564
           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
04565
           template<> struct ConwayPolynomial<601, 1> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<594»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 2> { using ZPZ = aerobus::zpz<601>; using type =
04566
      POLYV<ZPZV<1>, ZPZV<598>, ZPZV<7»; }; // NOLINT
            template<> struct ConwayPolynomial<601, 3> { using ZPZ = aerobus::zpz<601>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<594»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 4> { using ZPZ = aerobus::zpz<601>; using type =
04568
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<347>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 5> { using ZPZ = aerobus::zpz<601>; using type =
04569
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<594»; }; // NOLINT
           template<> struct ConwayPolynomial<601, 6> { using ZPZ = aerobus::zpz<601>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<128>, ZPZV<440>, ZPZV<49>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<601, 7> { using ZPZ = aerobus::zpz<601>; using type =
04571
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<594»; }; // NOLINT template<> struct ConwayPolynomial<601, 8> { using ZPZ = aerobus::zpz<601>; using type =
04572
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<550>, ZPZV<241>, ZPZV<490>, ZPZV<7»; }; //
           template<> struct ConwayPolynomial<601, 9> { using ZPZ = aerobus::zpz<601>; using type =
04573
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<487>, ZPZV<590>, ZPZV<594»;
       }; // NOLINT
04574
           template<> struct ConwayPolynomial<607, 1> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<604»; }; // NOLINT
04575
           template<> struct ConwayPolynomial<607, 2> { using ZPZ = aerobus::zpz<607>; using type =
       POLYV<ZPZV<1>, ZPZV<606>, ZPZV<3»; }; // NOLINT
04576
          template<> struct ConwayPolynomial<607, 3> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<604»; }; // NOLINT
template<> struct ConwayPolynomial<607, 4> { using ZPZ = aerobus::zpz<607>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<449>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<607, 5> { using ZPZ = aerobus::zpz<607>; using type =
04577
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<604»; }; // NOLINT
04579
           template<> struct ConwayPolynomial<607, 6> { using ZPZ = aerobus::zpz<607>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<10>, ZPZV<45>, ZPZV<478>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<607, 7> { using ZPZ = aerobus::zpz<607>; using type =
04580
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<604»; }; // NOLINT
           template<> struct ConwayPolynomial<607, 8> { using ZPZ = aerobus::zpz<607>; using type =
04581
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<468>, ZPZV<35>, ZPZV<449>, ZPZV<3*; };
       NOLINT
04582
           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
04583
           template<> struct ConwavPolynomial<613, 1> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<611»; }; // NOLINT
           template<> struct ConwayPolynomial<613, 2> { using ZPZ = aerobus::zpz<613>; using type =
       POLYV<ZPZV<1>, ZPZV<609>, ZPZV<2»; }; // NOLINT
04585
           template<> struct ConwayPolynomial<613, 3> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<611»; }; // NOLINT
  template<> struct ConwayPolynomial<613, 4> { using ZPZ = aerobus::zpz<613>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<333>, ZPZV<2»; }; // NOLINT</pre>
04586
           template<> struct ConwayPolynomial<613, 5> { using ZPZ = aerobus::zpz<613>; using type =
04587
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<32>, ZPZV<611»; }; // NOLINT
04588
           template<> struct ConwayPolynomial<613, 6> { using ZPZ = aerobus::zpz<613>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<60>, ZPZV<609>, ZPZV<595>, ZPZV<601>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<613, 7> { using ZPZ = aerobus::zpz<613>; using type
04589
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<61:»; };
           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
04591
      template<> struct ConwayPolynomial<613, 9> { using ZPZ = aerobus::zpz<613>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<513>, ZPZV<516>, ZPZV<611»;</pre>
       }; // NOLINT
04592
            template<> struct ConwayPolynomial<617, 1> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<614»; }; // NOLINT
04593
           template<> struct ConwayPolynomial<617, 2> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<612>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<617, 3> { using ZPZ = aerobus::zpz<617>; using type =
04594
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<614»; }; // NOLINT template<> struct ConwayPolynomial<617, 4> { using ZPZ = aerobus::zpz<617>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<503>, ZPZV<3»; }; // NOLINT
04596
           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 template<> struct ConwayPolynomial<617, 6> { using ZPZ = aerobus::zpz<617>; using type =
04597
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<318>, ZPZV<595>, ZPZV<310>, ZPZV<3»; }; // NOLINT
04598
           template<> struct ConwayPolynomial<617,
                                                          7> { using ZPZ = aerobus::zpz<617>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<614»; }; //
04599
           template<> struct ConwayPolynomial<617, 8> { using ZPZ = aerobus::zpz<617>; using type
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<501>, ZPZV<155>, ZPZV<13»; }; //
04600
           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»;
04601
          template<> struct ConwayPolynomial<619, 1> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<617»; }; // NOLINT
           template<> struct ConwayPolynomial<619, 2> { using ZPZ = aerobus::zpz<619>; using type =
04602
      POLYV<ZPZV<1>, ZPZV<618>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<619, 3> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<617»; }; // NOLINT template<> struct ConwayPolynomial<619, 4> { using ZPZ = aerobus::zpz<619>; using type =
04604
      template<> struct ConwayPolynomial<619, 5> { using ZPZ = aerobus::zpz<619>; using type =
04605
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<617»; }; // NOLINT
04606
           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»; }; // NOLIN
04607
          template<> struct ConwayPolynomial<619, 7> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<617»; }; // NOLIN template<> struct ConwayPolynomial<619, 8> { using ZPZ = aerobus::zpz<619>; using type =
04608
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0), ZPZV<10>, ZPZV<416>, ZPZV<383>, ZPZV<225>, ZPZV<2»; }; //
04609
          template<> struct ConwayPolynomial<619, 9> { using ZPZ = aerobus::zpz<619>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<510>, ZPZV<617»;
      }; // NOLINT
04610
          template<> struct ConwayPolynomial<631, 1> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<628»; }; // NOLINT
04611
           template<> struct ConwayPolynomial<631, 2> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<629>, ZPZV<3»; }; // NOLINT
           template<> struct ConwayPolynomial<631, 3> { using ZPZ = aerobus::zpz<631>; using type =
04612
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT template<> struct ConwayPolynomial<631, 4> { using ZPZ = aerobus::zpz<631>; using type =
04613
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<376>, ZPZV<376>, ZPZV<3»; }; // NOLINT

template<> struct ConwayPolynomial<631, 5> { using ZPZ = aerobus::zpz<631>; using type =
04614
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<628»; }; // NOLINT
           template<> struct ConwayPolynomial<631, 6> { using ZPZ = aerobus::zpz<631>; using type =
04615
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<516>, ZPZV<541>, ZPZV<106>, ZPZV<3»; }; // NOLINI
04616
           template<> struct ConwayPolynomial<631, 7> { using ZPZ = aerobus::zpz<631>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<628»; }; // NOLINT
           template<> struct ConwayPolynomial<631, 8> { using ZPZ = aerobus::zpz<631>; using type =
04617
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<379>, ZPZV<516>, ZPZV<187>, ZPZV<3»; }; //
          template<> struct ConwayPolynomial<631, 9> { using ZPZ = aerobus::zpz<631>; using type =
04618
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<296>, ZPZV<413>, ZPZV<628»;
      }; // NOLINT
           template<> struct ConwayPolynomial<641, 1> { using ZPZ = aerobus::zpz<641>; using type =
04619
      POLYV<ZPZV<1>, ZPZV<638»; }; // NOLINT
           template<> struct ConwayPolynomial<641, 2> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<635>, ZPZV<3»; }; // NOLINT
04621
          template<> struct ConwayPolynomial<641, 3> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<638»; }; // NOLINT
template<> struct ConwayPolynomial<641, 4> { using ZPZ = aerobus::zpz<641>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<641, 5> { using ZPZ = aerobus::zpz<641>; using type =
04622
04623
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<638»; }; // NOLINT
04624
           template<> struct ConwayPolynomial<641, 6> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<105>, ZPZV<557>, ZPZV<294>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<641, 7> { using ZPZ = aerobus::zpz<641>; using type :
04625
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<638»; }; // NOLINT
           template<> struct ConwayPolynomial<641, 8> { using ZPZ = aerobus::zpz<641>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<35, ZPZV<356>, ZPZV<392>, ZPZV<332>, ZPZV<3*; }; //
04627
           template<> struct ConwayPolynomial<641, 9> { using ZPZ = aerobus::zpz<641>; using type :
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<66>, ZPZV<668, ZPZV<638»;
      }; // NOLINT
04628
           template<> struct ConwayPolynomial<643, 1> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<632»; }; // NOLINT
04629
          template<> struct ConwayPolynomial<643, 2> { using ZPZ = aerobus::zpz<643>; using type =
      POLYV<ZPZV<1>, ZPZV<641>, ZPZV<11»; }; // NOLINT
           template<> struct ConwayPolynomial<643, 3> { using ZPZ = aerobus::zpz<643>; using type =
04630
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<632»; }; // NOLINT
          template<> struct ConwayPolynomial<643, 4> { using ZPZ = aerobus::zpz<643>; using type =
04631
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<60>, ZPZV<600>, ZPZV<11»; };
                                                                      // NOLINT
           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
04633
           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
                                                       7> { using ZPZ = aerobus::zpz<643>; using type
04634
           template<> struct ConwayPolynomial<643,
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<632»; }; //
          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
04636
      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<591>, ZPZV<591>, ZPZV<475>, ZPZV<632»;</pre>
      }; // NOLINT
    template<> struct ConwayPolynomial<647, 1> { using ZPZ = aerobus::zpz<647>; using type =
      POLYV<ZPZV<1>, ZPZV<642»; }; // NOLINT
          template<> struct ConwayPolynomial<647, 2> { using ZPZ = aerobus::zpz<647>; using type =
04638
      POLYV<ZPZV<1>, ZPZV<645>, ZPZV<5»; }; // NOLINT
          template<> struct ConwayPolynomial<647, 3> { using ZPZ = aerobus::zpz<647>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<642»; };
                                                                                       // NOLINT
         template<> struct ConwayPolynomial<647, 4> { using ZPZ = aerobus::zpz<647>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<643>, ZPZV<5»; }; // NOLINT
                template<> struct ConwayPolynomial<647, 5> { using ZPZ = aerobus::zpz<647>; using type =
04641
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<642»; }; // NOLINT
         template<> struct ConwayPolynomial<647, 6> { using ZPZ = aerobus::zpz<647>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<308>, ZPZV<385>, ZPZV<642>, ZPZV<5»; }; // NOLINT
04642
                template<> struct ConwayPolynomial<647, 7> { using ZPZ = aerobus::zpz<647>; using type
04643
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<642»; }; // NOLINT template<> struct ConwayPolynomial<647, 8> { using ZPZ = aerobus::zpz<647>; using type =
04644
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<603>, ZPZV<259>, ZPZV<271>, ZPZV<5»; }; //
         NOLINT
                template<> struct ConwayPolynomial<647, 9> { using ZPZ = aerobus::zpz<647>; using type =
04645
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<561>, ZPZV<562>, ZPZV<642»;
         }; // NOLINT
04646
                template<> struct ConwayPolynomial<653, 1> { using ZPZ = aerobus::zpz<653>; using type =
         POLYV<ZPZV<1>, ZPZV<651»; }; // NOLINT
                template<> struct ConwayPolynomial<653, 2> { using ZPZ = aerobus::zpz<653>; using type =
04647
         POLYV<ZPZV<1>, ZPZV<649>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<653, 3> { using ZPZ = aerobus::zpz<653>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<651»; }; // NOLINT template<> struct ConwayPolynomial<653, 4> { using ZPZ = aerobus::zpz<653>; using type =
04649
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<596>, ZPZV<2»; ); // NOLINT template<> struct ConwayPolynomial<653, 5> { using ZPZ = aerobus::zpz<653>; using type =
04650
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<651»; }; // NOLINT
                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<sup>*</sup>; }; // NOLINT
                template<> struct ConwayPolynomial<653, 7> { using ZPZ = aerobus::zpz<653>; using type =
04652
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<651»; };
04653
               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
                template<> struct ConwayPolynomial<653, 9> { using ZPZ = aerobus::zpz<653>; using type =
04654
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<651»;
         }; // NOLINT
04655
                template<> struct ConwayPolynomial<659, 1> { using ZPZ = aerobus::zpz<659>; using type =
         POLYV<ZPZV<1>, ZPZV<657»; }; // NOLINT
                template<> struct ConwayPolynomial<659, 2> { using ZPZ = aerobus::zpz<659>; using type =
         POLYV<ZPZV<1>, ZPZV<655>, ZPZV<2»; }; // NOLINT
                template<> struct ConwayPolynomial<659, 3> { using ZPZ = aerobus::zpz<659>; using type =
04657
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<657»; }; // NOLINT template<> struct ConwayPolynomial<659, 4> { using ZPZ = aerobus::zpz<659>; using type =
04658
         POLYY<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<351>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<659, 5> { using ZPZ = aerobus::zpz<659>; using type =
04659
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<657»; }; // NOLINT
04660
                template<> struct ConwayPolynomial<659, 6> { using ZPZ = aerobus::zpz<659>; using type =
          \verb"Polyv<2pzv<1>, & \verb"Zpzv<0>, & \verb"Zpzv<6>, & \verb"Zpzv<371>, & \verb"Zpzv<105>, & \verb"Zpzv<223>, & \verb"Zpzv<2»; & \verb"}; & // & \verb"Nolint" & \verb"Nolint
04661
               template<> struct ConwayPolynomial<659, 7> { using ZPZ = aerobus::zpz<659>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<657»; }; // NOLINT
               template<> struct ConwayPolynomial<659, 8> { using ZPZ = aerobus::zpz<659>; using type =
04662
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<358>, ZPZV<246>, ZPZV<90>, ZPZV<2»; }; //
04663
               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<0>, ZPZV<2>, ZPZV<592>, ZPZV<46>, ZPZV<657»;
         }; // NOLINT
04664
                template<> struct ConwayPolynomial<661, 1> { using ZPZ = aerobus::zpz<661>; using type =
         POLYV<ZPZV<1>, ZPZV<659»; }; // NOLINT
               template<> struct ConwayPolynomial<661, 2> { using ZPZ = aerobus::zpz<661>; using type =
         POLYV<ZPZV<1>, ZPZV<660>, ZPZV<2»; }; // NOLINT
04666
                template<> struct ConwayPolynomial<661, 3> { using ZPZ = aerobus::zpz<661>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<659»; ); // NOLINT
template<> struct ConwayPolynomial<661, 4> { using ZPZ = aerobus::zpz<661>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<616>, ZPZV<2»; }; // NOLINT
04667
                 template<> struct ConwayPolynomial<661, 5> { using ZPZ = aerobus::zpz<661>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<659»; }; // NOLINT
04669
               template<> struct ConwayPolynomial<661, 6> { using ZPZ = aerobus::zpz<661>; using type =
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<456>, ZPZV<382>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<661, 7> { using ZPZ = aerobus::zpz<661>; using type =
04670
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<659»; }; // NOLINT
                template<> struct ConwayPolynomial<661, 8> { using ZPZ = aerobus::zpz<661>; using type
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<612>, ZPZV<285>, ZPZV<72>, ZPZV<72>; };
         template<> struct ConwayPolynomial<661, 9> { using ZPZ = aerobus::zpz<661>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<389>, ZPZV<389>, ZPZV<220>, ZPZV<659»;</pre>
04672
         }; // NOLINT
                template<> struct ConwayPolynomial<673, 1> { using ZPZ = aerobus::zpz<673>; using type =
         POLYV<ZPZV<1>, ZPZV<668»; }; // NOLINT
04674
                template<> struct ConwayPolynomial<673, 2> { using ZPZ = aerobus::zpz<673>; using type =
         POLYV<ZPZV<1>, ZPZV<672>, ZPZV<5»; }; // NOLINT
                template<> struct ConwayPolynomial<673, 3> { using ZPZ = aerobus::zpz<673>; using type =
04675
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<668»; }; // NOLINT template<> struct ConwayPolynomial<673, 4> { using ZPZ = aerobus::zpz<673>; using type =
04676
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<416>, ZPZV<5»; }; // NOLINT
04677
               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 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
04678
```

```
template<> struct ConwayPolynomial<673, 7> { using ZPZ = aerobus::zpz<673>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6
, ZPZV
, Z
04680
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<66>, ZPZV<587>, ZPZV<302>, ZPZV<5»; }; //
           NOLINT
                  template<> struct ConwayPolynomial<673, 9> { using ZPZ = aerobus::zpz<673>; using type =
04681
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<347>, ZPZV<553>, ZPZV<668»;
           }; // NOLINT
                 template<> struct ConwayPolynomial<677, 1> { using ZPZ = aerobus::zpz<677>; using type =
04682
          POLYV<ZPZV<1>, ZPZV<675»; }; // NOLINT
                  template<> struct ConwayPolynomial<677, 2> { using ZPZ = aerobus::zpz<677>; using type =
04683
          POLYV<ZPZV<1>, ZPZV<672>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<677, 3> { using ZPZ = aerobus::zpz<677>; using type =
04684
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<675»; }; // NOLINT
04685
                 template<> struct ConwayPolynomial<677, 4> { using ZPZ = aerobus::zpz<677>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<631>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<677, 5> { using ZPZ = aerobus::zpz<677>; using type =
04686
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<675s; }; // NOLINT template<> struct ConwayPolynomial<677, 6> { using ZPZ = aerobus::zpz<677>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<446>, ZPZV<632>, ZPZV<50>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<677, 7> { using ZPZ = aerobus::zpz<677>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<10>, ZPZV<675»; };
04689
                 template<> struct ConwayPolynomial<677, 8> { using ZPZ = aerobus::zpz<677>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<619>, ZPZV<152>, ZPZV<2*; }; //
          NOLINT
                  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
04691
                  template<> struct ConwayPolynomial<683, 1> { using ZPZ = aerobus::zpz<683>; using type =
          POLYV<ZPZV<1>, ZPZV<678»; }; // NOLINT
                 template<> struct ConwayPolynomial<683, 2> { using ZPZ = aerobus::zpz<683>; using type =
04692
          POLYV<ZPZV<1>, ZPZV<682>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<683, 3> { using ZPZ = aerobus::zpz<683>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<678»; }; // NOLINT
                  template<> struct ConwayPolynomial<683, 4> { using ZPZ = aerobus::zpz<683>; using type =
04694
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<455>, ZPZV<5»; }; // NOLINT
                  template<> struct ConwayPolynomial<683, 5> { using ZPZ = aerobus::zpz<683>; using type =
04695
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<678»; }; // NOLINT
                  template<> struct ConwayPolynomial<683, 6> { using ZPZ = aerobus::zpz<683>; using type =
04696
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<644>, ZPZV<109>, ZPZV<434>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<683, 7> { using ZPZ = aerobus::zpz<683>; using type =
04697
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<68*; }; // NOLINT template<> struct ConwayPolynomial<683, 8> { using ZPZ = aerobus::zpz<683>; using type =
04698
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<383>, ZPZV<184>, ZPZV<65>, ZPZV<65»; }; //
04699
                  template<> struct ConwayPolynomial<683, 9> { using ZPZ = aerobus::zpz<683>; using type
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0
           }; // NOLINT
04700
                  template<> struct ConwavPolynomial<691, 1> { using ZPZ = aerobus::zpz<691>; using type =
          POLYV<ZPZV<1>, ZPZV<688»; }; // NOLINT
                  template<> struct ConwayPolynomial<691, 2> { using ZPZ = aerobus::zpz<691>; using type =
          POLYV<ZPZV<1>, ZPZV<686>, ZPZV<3»; }; // NOLINT
04702
                 template<> struct ConwayPolynomial<691, 3> { using ZPZ = aerobus::zpz<691>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<688»; }; // NOLINT

template<> struct ConwayPolynomial<691, 4> { using ZPZ = aerobus::zpz<691>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<632>, ZPZV<3»; }; // NOLINT

template<> struct ConwayPolynomial<691, 5> { using ZPZ = aerobus::zpz<691>; using type =
04703
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688»; }; // NOLINT
                  template<> struct ConwayPolynomial<691, 6> { using ZPZ = aerobus::zpz<691>; using type =
04705
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<579>, ZPZV<408>, ZPZV<262>, ZPZV<3»; }; // NOLINT
04706
                 template<> struct ConwayPolynomial<691, 7> { using ZPZ = aerobus::zpz<691>; using type =
          POLYV-ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<688%; }; // NOLINT
                  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<32); //
          template<> struct ConwayPolynomial<691, 9> { using ZPZ = aerobus::zpz<691>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<443>, ZPZV<688»;</pre>
04708
           }; // NOLINT
                  template<> struct ConwayPolynomial<701, 1> { using ZPZ = aerobus::zpz<701>; using type =
04709
          POLYV<ZPZV<1>, ZPZV<699»; }; // NOLINT
04710
                  template<> struct ConwayPolynomial<701, 2> { using ZPZ = aerobus::zpz<701>; using type =
          POLYV<ZPZV<1>, ZPZV<697>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<701, 3> { using ZPZ = aerobus::zpz<701>; using type =
04711
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<699»; }; // NOLINT template<> struct ConwayPolynomial<701, 4> { using ZPZ = aerobus::zpz<701>; using type =
04712
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<379>, ZPZV<2»; }; // NOLINT
                 template<> struct ConwayPolynomial<701, 5> { using ZPZ = aerobus::zpz<701>; using type =
04713
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<699»; }; // NOLINT
                 template<> struct ConwayPolynomial<701, 6> { using ZPZ = aerobus::zpz<701>; using type =
04714
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<571>, ZPZV<327>, ZPZV<327>, ZPZV<285>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial</ri>
04715
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<699»; };
                 template<> struct ConwayPolynomial<701, 8> { using ZPZ = aerobus::zpz<701>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<619>, ZPZV<206>, ZPZV<593>, ZPZV<593>, ZPZV<2»; }; //
          template<> struct ConwayPolynomial<701, 9> { using ZPZ = aerobus::zpz<701>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<45>, ZPZV<45>, ZPZV<459>, ZPZV<373>, ZPZV<699»;</pre>
```

```
}; // NOLINT
04718
            template<> struct ConwayPolynomial<709, 1> { using ZPZ = aerobus::zpz<709>; using type =
       POLYV<ZPZV<1>, ZPZV<707»; }; // NOLINT
            template<> struct ConwayPolynomial<709, 2> { using ZPZ = aerobus::zpz<709>; using type =
04719
       POLYV<ZPZV<1>, ZPZV<705>, ZPZV<2»; }; // NOLINT
            template<> struct ConwayPolynomial<709, 3> { using ZPZ = aerobus::zpz<709>; using type =
04720
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<707»; }; // NOLINT template<> struct ConwayPolynomial<709, 4> { using ZPZ = aerobus::zpz<709>; using type =
04721
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<384>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<709, 5> { using ZPZ = aerobus::zpz<709>; using type =
04722
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<707»; }; // NOLINT
      template<> struct ConwayPolynomial<709, 6> { using ZPZ = aerobus::zpz<709>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<669>, ZPZV<514>, ZPZV<295>, ZPZV<2»; }; // NOLINT
04723
            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»; }; //
04725
           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
            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<0>, ZPZV<25>, ZPZV<257>, ZPZV<257>, ZPZV<171>, ZPZV<707»;
       }; // NOLINT
04727
            template<> struct ConwayPolynomial<719, 1> { using ZPZ = aerobus::zpz<719>; using type =
       POLYV<ZPZV<1>, ZPZV<708»; }; // NOLINT
           template<> struct ConwayPolynomial<719, 2> { using ZPZ = aerobus::zpz<719>; using type =
04728
       POLYV<ZPZV<1>, ZPZV<715>, ZPZV<11»; }; // NOLINT
            template<> struct ConwayPolynomial<719, 3> { using ZPZ = aerobus::zpz<719>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
            template<> struct ConwayPolynomial<719, 4> { using ZPZ = aerobus::zpz<719>; using type =
04730
        \verb"POLYV<ZPZV<1>, \ \verb"ZPZV<0>, \ \verb"ZPZV<5>, \ \verb"ZPZV<602>, \ \verb"ZPZV<11"; \ \verb"}; \ \ // \ \verb"NOLINT" 
           template<> struct ConwayPolynomial<719, 5> { using ZPZ = aerobus::zpz<719>; using type =
04731
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<708»; }; // NOLINT
04732
            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
04733
           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<0>, ZPZV<11>, ZPZV<708»; }; // NOLINT template<> struct ConwayPolynomial<719, 8> { using ZPZ = aerobus::zpz<719>; using type =
04734
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<714>, ZPZV<362>, ZPZV<244>, ZPZV<11»; }; //
            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
04736
            template<> struct ConwayPolynomial<727, 1> { using ZPZ = aerobus::zpz<727>; using type =
       POLYV<ZPZV<1>, ZPZV<722»; }; // NOLINT
            template<> struct ConwayPolynomial<727, 2> { using ZPZ = aerobus::zpz<727>; using type =
04737
       POLYV<ZPZV<1>, ZPZV<725>, ZPZV<5»; }; // NOLINT
04738
            template<> struct ConwayPolynomial<727, 3> { using ZPZ = aerobus::zpz<727>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<72), '; // NOLINT template<> struct ConwayPolynomial<727, 4> { using ZPZ = aerobus::zpz<727>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<723>, ZPZV<5>; // NOLINT template<> struct ConwayPolynomial<727, 5> { using ZPZ = aerobus::zpz<727>; using type =
04739
04740
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<722»; }; // NOLINT
            template<> struct ConwayPolynomial<727, 6> { using ZPZ = aerobus::zpz<727>; using type =
04741
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<86>, ZPZV<397>, ZPZV<672>, ZPZV<5»; }; // NOLINT
      template<> struct ConwayPolynomial<727, 7> { using ZPZ = aerobus::2pz<727>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<17>, ZPZV<722»; }; // NOLINT</pre>
04742
           template<> struct ConwayPolynomial<727, 8> { using ZPZ = aerobus::zpz<727>; using type =
04743
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<63>, ZPZV<671>, ZPZV<368>, ZPZV<5»; }; //
       template<> struct ConwayPolynomial<727, 9> { using ZPZ = aerobus::zpz<727>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<573>, ZPZV<573>, ZPZV<502>, ZPZV<722»;
       }; // NOLINT
            template<> struct ConwayPolynomial<733, 1> { using ZPZ = aerobus::zpz<733>; using type =
04745
       POLYV<ZPZV<1>, ZPZV<727»; }; // NOLINT
            template<> struct ConwayPolynomial<733, 2> { using ZPZ = aerobus::zpz<733>; using type =
       POLYV<ZPZV<1>, ZPZV<732>, ZPZV<6»; }; // NOLINT
04747
           template<> struct ConwayPolynomial<733, 3> { using ZPZ = aerobus::zpz<733>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<727»; }; // NOLINT template<> struct ConwayPolynomial<733, 4> { using ZPZ = aerobus::zpz<733>; using type =
04748
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<539, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<733, 5> { using ZPZ = aerobus::zpz<733>; using type =
04749
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<727»; }; // NOLINT
04750
           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<66; }; // NOLINT template<> struct ConwayPolynomial<733, 7> { using ZPZ = aerobus::zpz<733>; using type =
04751
       POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<727»; };
                                                                                                           // NOLINT
            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»; };
04753
           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<0>, ZPZV<2>, ZPZV<337>, ZPZV<6>, ZPZV<727»; };
       // NOLINT
            template<> struct ConwayPolynomial<739, 1> { using ZPZ = aerobus::zpz<739>; using type =
       POLYV<ZPZV<1>, ZPZV<736»; }; // NOLINT
04755
           template<> struct ConwayPolynomial<739, 2> { using ZPZ = aerobus::zpz<739>; using type =
      POLYV<ZPZV<1>, ZPZV<733>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<739, 3> { using ZPZ = aerobus::zpz<739>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<736»; }; // NOLINT
04756
```

```
04757
           template<> struct ConwayPolynomial<739, 4> { using ZPZ = aerobus::zpz<739>; using type =
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<3>, ZPZV<678>, ZPZV<63*, ; // NOLINT template<> struct ConwayPolynomial<739, 5> { using ZPZ = aerobus::zpz<739>; using type =
04758
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<736»; }; // NOLINT
      template<> struct ConwayPolynomial<739, 6> { using ZPZ = aerobus::zpz<739>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<42>, ZPZV<447>, ZPZV<625>, ZPZV<3»; }; // NOLINT
04759
           template<> struct ConwayPolynomial<739, 7> { using ZPZ = aerobus::zpz<739>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<44>, ZPZV<736»; };
04761
          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
04762
           template<> struct ConwayPolynomial<739, 9> { using ZPZ = aerobus::zpz<739>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<1>, ZPZV<6166, ZPZV<81, ZPZV<736%;
       }; // NOLINT
04763
           template<> struct ConwayPolynomial<743, 1> { using ZPZ = aerobus::zpz<743>; using type =
      POLYY<ZPZV<1>, ZPZV<738»; }; // NOLINT template<> struct ConwayPolynomial<743, 2> { using ZPZ = aerobus::zpz<743>; using type =
04764
      POLYV<ZPZV<1>, ZPZV<742>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<743, 3> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<738»; }; // NOLINT template<> struct ConwayPolynomial<743, 4> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<425>, ZPZV<5»; }; // NOLINT
04767
           template<> struct ConwayPolynomial<743, 5> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<738»; }; // NOLINT
           template<> struct ConwayPolynomial<743, 6> { using ZPZ = aerobus::zpz<743>; using type =
04768
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<236>, ZPZV<471>, ZPZV<88>, ZPZV<5»; }; // NOLINT
04769
           template<> struct ConwayPolynomial<743, 7> { using ZPZ = aerobus::zpz<743>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<738»; }; // NOLINT template<> struct ConwayPolynomial<743, 8> { using ZPZ = aerobus::zpz<743>; using type =
04770
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<551>, ZPZV<279>, ZPZV<588>, ZPZV<5»; }; //
      NOLINT
04771
           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<3>, ZPZV<327>, ZPZV<676>, ZPZV<738»;
      }; // NOLINT
04772
           template<> struct ConwayPolynomial<751, 1> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<748»; }; // NOLINT
           template<> struct ConwayPolynomial<751, 2> { using ZPZ = aerobus::zpz<751>; using type =
04773
      POLYV<ZPZV<1>, ZPZV<749>, ZPZV<3»; }; // NOLINT
04774
           template<> struct ConwayPolynomial<751, 3> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<748»; }; // NOLINT template<> struct ConwayPolynomial<751, 4> { using ZPZ = aerobus::zpz<751>; using type =
04775
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<525>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<751, 5> { using ZPZ = aerobus::zpz<751>; using type =
04776
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<748»; }; // NOLINT
           template<> struct ConwayPolynomial<751, 6> { using ZPZ = aerobus::zpz<751>; using type =
      POLYV<2PZV<1>, 2PZV<0>, 2PZV<2>, 2PZV<298>, ZPZV<633>, ZPZV<539>, ZPZV<3»; }; // NOLINT
04778
           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<7>, ZPZV<748»; }; // NOLINT
           template<> struct ConwayPolynomial<751, 8> { using ZPZ = aerobus::zpz<751>; using type =
04779
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<41>, ZPZV<243>, ZPZV<672>, ZPZV<3»; }; //
      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»;
04780
       }; // NOLINT
04781
           template<> struct ConwayPolynomial<757, 1> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 2> { using ZPZ = aerobus::zpz<757>; using type =
      POLYV<ZPZV<1>, ZPZV<753>, ZPZV<2»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 3> { using ZPZ = aerobus::zpz<757>; using type =
04783
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<755»; }; // NOLINT template<> struct ConwayPolynomial<757, 4> { using ZPZ = aerobus::zpz<757>; using type =
04784
      POLYV<2PZV<1>, ZPZV<0>, ZPZV<10>, ZPZV<537, ZPZV<2*; }; // NOLINT template<> struct ConwayPolynomial<757, 5> { using ZPZ = aerobus::zpz<757>; using type =
04785
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<13>, ZPZV<755»; }; // NOLINT
04786
           template<> struct ConwayPolynomial<757, 6> { using ZPZ = aerobus::zpz<757>; using type =
      POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<753>, ZPZV<739>, ZPZV<745>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<757, 7> { using ZPZ = aerobus::zpz<757>; using type =
04787
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<757, 8> { using ZPZ = aerobus::zpz<757>; using type =
04788
       POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<494>, ZPZV<4110>, ZPZV<509>, ZPZV<2»; };
04789
          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<608>, ZPZV<702>, ZPZV<755»;
       }; // NOLINT
04790
           template<> struct ConwayPolynomial<761, 1> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<755»; }; // NOLINT
           template<> struct ConwayPolynomial<761, 2> { using ZPZ = aerobus::zpz<761>; using type =
04791
      POLYV<ZPZV<1>, ZPZV<758>, ZPZV<6»; }; // NOLINT
04792
           template<> struct ConwayPolynomial<761, 3> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<12>, ZPZV<755»; }; // NOLINT
template<> struct ConwayPolynomial<761, 4> { using ZPZ = aerobus::zpz<761>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<658>, ZPZV<6»; }; // NOLINT
04793
           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
04795
          template<> struct ConwayPolynomial<761, 6> { using ZPZ = aerobus::zpz<761>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<634>, ZPZV<597>, ZPZV<155>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<761, 7> { using ZPZ = aerobus::zpz<761>; using type =
```

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<755»; };
              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<54»; }; //
        NOLINT
04798
              template<> struct ConwayPolynomial<761, 9> { using ZPZ = aerobus::zpz<761>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<317>, ZPZV<571>, ZPZV<755»;
        }; // NOLINT
04799
              template<> struct ConwayPolynomial<769, 1> { using ZPZ = aerobus::zpz<769>; using type =
        POLYV<ZPZV<1>, ZPZV<758»; }; // NOLINT
             template<> struct ConwayPolynomial<769, 2> { using ZPZ = aerobus::zpz<769>; using type =
04800
        POLYV<ZPZV<1>, ZPZV<765>, ZPZV<11»; }; // NOLINT
              template<> struct ConwayPolynomial<769, 3> { using ZPZ = aerobus::zpz<769>; using type =
04801
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<758»; }; // NOLINT
               template<> struct ConwayPolynomial<?769, 4> { using ZPZ = aerobus::zpz<769>; using type =
04802
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<32>, ZPZV<741>, ZPZV<11»; }; // NOLINT
        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</pre>
04803
              template<> struct ConwayPolynomial<769, 6> { using ZPZ = aerobus::zpz<769>; using type =
04804
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<43>, ZPZV<326>, ZPZV<650>, ZPZV<11»; }; // NOLINT
04805
              template<> struct ConwayPolynomial<769, 7> { using ZPZ = aerobus::zpz<769>; using type
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<758»; }; //
04806
              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
04807
              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<623>, ZPZV<751>, ZPZV<758»;
04808
              template<> struct ConwayPolynomial<773, 1> { using ZPZ = aerobus::zpz<773>; using type =
        POLYV<ZPZV<1>, ZPZV<771»; }; // NOLINT
             template<> struct ConwayPolynomial<773, 2> { using ZPZ = aerobus::zpz<773>; using type =
04809
        POLYV<ZPZV<1>, ZPZV<772>, ZPZV<2»; }; // NOLINT
04810
              template<> struct ConwayPolynomial<773, 3> { using ZPZ = aerobus::zpz<773>; using type =
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<2>, ZPZY<771%; }; // NOLINT template<> struct ConwayPolynomial<773, 4> { using ZPZ = aerobus::zpz<773>; using type =
04811
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<444>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<773, 5> { using ZPZ = aerobus::zpz<773>; using type =
04812
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<771»; }; // NOLINT 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»; };
             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
04815
             template<> struct ConwayPolynomial<773, 8> { using ZPZ = aerobus::zpz<773>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<484>, ZPZV<94>, ZPZV<693>, ZPZV<2»; };
              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
04817
              template<> struct ConwayPolynomial<787, 1> { using ZPZ = aerobus::zpz<787>; using type =
        POLYV<ZPZV<1>, ZPZV<785»; }; // NOLINT
              template<> struct ConwayPolynomial<787, 2> { using ZPZ = aerobus::zpz<787>; using type =
04818
        POLYV<ZPZV<1>, ZPZV<786>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<787, 3> { using ZPZ = aerobus::zpz<787>; using type =
04819
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<785»; }; // NOLINT template<> struct ConwayPolynomial<787, 4> { using ZPZ = aerobus::zpz<787>; using type =
04820
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<3>, ZPZV<2; ; ; // NOLINT
template<> struct ConwayPolynomial<787, 5> { using ZPZ = aerobus::zpz<787>; using type =
04821
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<785»; }; // NOLINT
              template<> struct ConwayPolynomial<787, 6> { using ZPZ = aerobus::zpz<787>; using type =
04822
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<98>, ZPZV<512>, ZPZV<606>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<787, 7> { using ZPZ = aerobus::zpz<787>; using type =
04823
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<785»; }; // NOLINT template<> struct ConwayPolynomial<787, 8> { using ZPZ = aerobus::zpz<787>; using type =
04824
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<612>, ZPZV<26>, ZPZV<715>, ZPZV<2»; };
04825
             template<> struct ConwayPolynomial<787, 9> { using ZPZ = aerobus::zpz<787>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<480>, ZPZV<573>, ZPZV<785»;
         }; // NOLINT
              \texttt{template<> struct ConwayPolynomial<797, 1> \{ using ZPZ = aerobus:: zpz<797>; using type = 200 t
04826
        POLYV<ZPZV<1>, ZPZV<795»; }; // NOLINT
04827
               template<> struct ConwayPolynomial<797, 2> { using ZPZ = aerobus::zpz<797>; using type =
        POLYV<ZPZV<1>, ZPZV<793>, ZPZV<2»; }; // NOLINT
04828
             template<> struct ConwayPolynomial<797, 3> { using ZPZ = aerobus::zpz<797>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<795»; }; // NOLINT template<> struct ConwayPolynomial<797, 4> { using ZPZ = aerobus::zpz<797>; using type =
04829
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<717>, ZPZV<2*; }; // NOLINT 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
04831
              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
              template<> struct ConwayPolynomial<797, 7> { using ZPZ = aerobus::zpz<797>; using type
04832
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<11>, ZPZV<795»; }; // NOLINT
              template<> struct ConwayPolynomial<797, 8> { using ZPZ = aerobus::zpz<797>; using type =
04833
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<596>, ZPZV<747>, ZPZV<389>, ZPZV<2»; };
        NOLINT
04834
             POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<240>, ZPZV<599>, ZPZV<795»;
        }; // NOLINT
```

```
04835
               template<> struct ConwayPolynomial<809, 1> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<806»; }; // NOLINT
04836
              template<> struct ConwayPolynomial<809, 2> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<799>, ZPZV<3»; }; // NOLINT
04837
              template<> struct ConwayPolynomial<809, 3> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<806»; ); // NOLINT template<> struct ConwayPolynomial<809, 4> { using ZPZ = aerobus::zpz<809>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<644>, ZPZV<3»; }; // NOLINT
04839
              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
              template<> struct ConwayPolynomial<809, 6> { using ZPZ = aerobus::zpz<809>; using type =
04840
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<562>, ZPZV<75>, ZPZV<43>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<809, 7> { using ZPZ = aerobus::zpz<809>; using type
04841
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<806»; }; //
04842
             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<673>, ZPZV<3»; }; //
        NOLTNT
        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<341>, ZPZV<341>, ZPZV<727>, ZPZV<806»;
04843
              template<> struct ConwayPolynomial<811, 1> { using ZPZ = aerobus::zpz<811>; using type =
        POLYV<ZPZV<1>, ZPZV<808»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 2> { using ZPZ = aerobus::zpz<811>; using type =
04845
        POLYV<ZPZV<1>, ZPZV<806>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<811, 3> { using ZPZ = aerobus::zpz<811>; using type =
04846
        POLYY<ZPZY<1>, ZPZY<0>, ZPZY<1>, ZPZY<108; }; // NOLINT template<> struct ConwayPolynomial<811, 4> { using ZPZ = aerobus::zpz<811>; using type =
04847
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<453>, ZPZV<3»; }; // NOLINT
04848
               template<> struct ConwayPolynomial<811, 5> { using ZPZ = aerobus::zpz<811>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<388*; }; // NOLINT template<> struct ConwayPolynomial<811, 6> { using ZPZ = aerobus::zpz<811>; using type =
04849
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<780>, ZPZV<755>, ZPZV<307>, ZPZV<3»; }; // NOLINT
              template<> struct ConwayPolynomial<811, 7> { using ZPZ = aerobus::zpz<811>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<808*; }; // NOLINT
              template<> struct ConwayPolynomial<811, 8> { using ZPZ = aerobus::zpz<811>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<663>, ZPZV<806>, ZPZV<525>, ZPZV<3»; }; //
        NOLINT
              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<300>, ZPZV<808»;
        }; // NOLINT
04853
              template<> struct ConwayPolynomial<821, 1> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<819»; }; // NOLINT
              template<> struct ConwayPolynomial<821, 2> { using ZPZ = aerobus::zpz<821>; using type =
04854
        POLYV<ZPZV<1>, ZPZV<816>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<821, 3> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<819»; }; // NOLINT
04856
              template<> struct ConwayPolynomial<821, 4> { using ZPZ = aerobus::zpz<821>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<15>, ZPZV<662>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<821, 5> { using ZPZ = aerobus::zpz<821>; using type =
04857
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<819»; // NOLINT
04858
              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»; }; // NOLIN
04859
              template<> struct ConwayPolynomial<821, 7> { using ZPZ = aerobus::zpz<821>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<819»; }; // NOLI template<> struct ConwayPolynomial<821, 8> { using ZPZ = aerobus::zpz<821>; using type =
                                                                                                                                      // NOLTNT
04860
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<556>, ZPZV<589>, ZPZV<2»; }; //
04861
              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
04862
              template<> struct ConwayPolynomial<823, 1> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<820»; }; // NOLINT
04863
               template<> struct ConwayPolynomial<823, 2> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<3»; }; // NOLINT
04864
              template<> struct ConwayPolynomial<823, 3> { using ZPZ = aerobus::zpz<823>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
template<> struct ConwayPolynomial<823, 4> { using ZPZ = aerobus::zpz<823>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<819>, ZPZV<3»; }; // NOLINT
template<> struct ConwayPolynomial<823, 5> { using ZPZ = aerobus::zpz<823>; using type =
04865
04866
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<820»; }; // NOLINT
              template<> struct ConwayPolynomial<823, 6> { using ZPZ = aerobus::zpz<823>; using type =
04867
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<822>, ZPZV<616>, ZPZV<744>, ZPZV<3»; }; // NOLINT
04868
              template<> struct ConwayPolynomial<823, 7> { using ZPZ = aerobus::zpz<823>; using type
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<20>, ZPZV<20>, ZPZV<20>, ZPZV<20>; ZPZV<20>
04869
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<451>, ZPZV<437>, ZPZV<31>, ZPZV<31>; };
        NOLINT
04870
              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<60, ZPZV<740>, ZPZV<609>, ZPZV<820»;
        }; // NOLINT
04871
               template<> struct ConwayPolynomial<827, 1> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<825»; }; // NOLINT
              template<> struct ConwayPolynomial<827, 2> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<821>, ZPZV<2»; }; // NOLINT
             template<> struct ConwayPolynomial<827, 3> { using ZPZ = aerobus::zpz<827>; using type =
04873
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<825»; }; // NOLINT template<> struct ConwayPolynomial<827, 4> { using ZPZ = aerobus::zpz<827>; using type =
```

9.3 aerobus.h 169

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<18>, ZPZV<605>, ZPZV<2»; };
               template<> struct ConwayPolynomial<827, 5> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<825»; }; // NOLINT
04876
              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 template<> struct ConwayPolynomial<827, 7> { using ZPZ = aerobus::zpz<827>; using type
04877
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<825»; }; //
04878
               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»; };
04879
              template<> struct ConwayPolynomial<827, 9> { using ZPZ = aerobus::zpz<827>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>77>, ZPZV<372>, ZPZV<825»;
        }; // NOLINT
  template<> struct ConwayPolynomial<829, 1> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<827»; }; // NOLINT
04881
              template<> struct ConwayPolynomial<829, 2> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<828>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<829, 3> { using ZPZ = aerobus::zpz<829>; using type =
04882
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<827»; }; // NOLINT
              template<> struct ConwayPolynomial<829, 4> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<9>, ZPZV<604>, ZPZV<2»; }; // NOLINT
04884
              template<> struct ConwayPolynomial<829, 5> { using ZPZ = aerobus::zpz<829>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<827»; }; // NOLINT template<> struct ConwayPolynomial<829, 6> { using ZPZ = aerobus::zpz<829>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<347<, ZPZV<817>, ZPZV<2»; }; // NOLINT
04885
               template<> struct ConwayPolynomial<829, 7> { using ZPZ = aerobus::zpz<829>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<827»; };
04887
              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
              template<> struct ConwayPolynomial<829, 9> { using ZPZ = aerobus::zpz<829>; using type =
04888
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<621>, ZPZV<552>, ZPZV<827»;
         }; // NOLINT
              template<> struct ConwayPolynomial<839, 1> { using ZPZ = aerobus::zpz<839>; using type =
04889
        POLYV<ZPZV<1>, ZPZV<828»; }; // NOLINT
               template<> struct ConwayPolynomial<839, 2> { using ZPZ = aerobus::zpz<839>; using type =
04890
        POLYV<ZPZV<1>, ZPZV<838>, ZPZV<11»; }; // NOLINT
               template<> struct ConwayPolynomial<839, 3> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<828»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 4> { using ZPZ = aerobus::zpz<839>; using type =
04892
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<609>, ZPZV<11»; }; // NOLINT
              template<> struct ConwayPolynomial<839, 5> { using ZPZ = aerobus::zpz<839>; using type =
04893
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<828»; }; // NOLINT
04894
              template<> struct ConwayPolynomial<839, 6> { using ZPZ = aerobus::zpz<839>; using type =
        POLYV<2PZV<1>, 2PZV<0>, ZPZV<1>, ZPZV<370>, ZPZV<537>, ZPZV<23>, ZPZV<11»; }; // NOLINT
04895
              template<> struct ConwayPolynomial<839, 7> { using ZPZ = aerobus::zpz<839>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<828%; }; // NOLINT template<> struct ConwayPolynomial<839, 8> { using ZPZ = aerobus::zpz<839>; using type =
04896
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<16>, ZPZV<553>, ZPZV<779>, ZPZV<329>, ZPZV<11»; }; //
         NOLINT
04897
              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<0>, ZPZV<0>, ZPZV<349>, ZPZV<206>, ZPZV<828*;
         }; // NOLINT
04898
              template<> struct ConwayPolynomial<853, 1> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<851»; }; // NOLINT
               template<> struct ConwayPolynomial<853, 2> { using ZPZ = aerobus::zpz<853>; using type =
04899
         POLYV<ZPZV<1>, ZPZV<852>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<853, 3> { using ZPZ = aerobus::zpz<853>; using type =
04900
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 4> { using ZPZ = aerobus::zpz<853>; using type =
04901
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<623>, ZPZV<2»; }; // NOLINT
template<> struct ConwayPolynomial<853, 5> { using ZPZ = aerobus::zpz<853>; using type =
04902
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<851»; }; // NOLINT
               template<> struct ConwayPolynomial<853, 6> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<2PZV<1>, 2PZV<0>, 2PZV<0>, 2PZV<276>, ZPZV<194>, ZPZV<512>, ZPZV<2»; }; // NOLINT
04904
              template<> struct ConwayPolynomial<853, 7> { using ZPZ = aerobus::zpz<853>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<45, ZPZV<851»; }; // NOLINT template<> struct ConwayPolynomial<853, 8> { using ZPZ = aerobus::zpz<853>; using type =
04905
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<544>, ZPZV<846>, ZPZV<118>, ZPZV<118>, ZPZV<2»; }; //
         NOLINT
              template<> struct ConwayPolynomial<853, 9> { using ZPZ = aerobus::zpz<853>; using type =
04906
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<677>, ZPZV<821>, ZPZV<851»;
         }; // NOLINT
04907
               template<> struct ConwayPolynomial<857, 1> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<854»; }; // NOLINT
               template<> struct ConwayPolynomial<857, 2> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<850>, ZPZV<3»; }; // NOLINT
04909
              template<> struct ConwayPolynomial<857, 3> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<854»; }; // NOLINT template<> struct ConwayPolynomial<857, 4> { using ZPZ = aerobus::zpz<857>; using type =
04910
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<528>, 
04911
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<854»; }; // NOLINT
04912
              template<> struct ConwayPolynomial<857, 6> { using ZPZ = aerobus::zpz<857>; using type =
        POLYV<ZPZV<1>, ZPZV<0, ZPZV<1>, ZPZV<32>, ZPZV<824>, ZPZV<65>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<857, 7> { using ZPZ = aerobus::zpz<857>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<854»; }; // NOLINT
04913
```

170 File Documentation

```
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
        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<30>, ZPZV<308>, ZPZV<719>, ZPZV<854»;</pre>
04915
         }; // NOLINT
04916
               template<> struct ConwayPolynomial<859, 1> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<ZPZV<1>, ZPZV<857»; }; // NOLINT
04917
              template<> struct ConwayPolynomial<859, 2> { using ZPZ = aerobus::zpz<859>; using type =
        POLYV<ZPZV<1>, ZPZV<858>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<859, 3> { using ZPZ = aerobus::zpz<859>; using type =
04918
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 4> { using ZPZ = aerobus::zpz<859>; using type =
04919
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<530>, ZPZV<2»; }; // NOLINT
04920
              template<> struct ConwayPolynomial<859, 5> { using ZPZ = aerobus::zpz<859>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<12>, ZPZV<857»; }; // NOLINT template<> struct ConwayPolynomial<859, 6> { using ZPZ = aerobus::zpz<859>; using type =
04921
        POLYYCZPZVc1>, ZPZVcO>, ZPZVcO>, ZPZVc419>, ZPZVc466>, ZPZVc566>, ZPZVc2>; ; // NOLINT template<> struct ConwayPolynomial<859, 7> { using ZPZ = aerobus::zpzc859>; using type
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<857»; };
               template<> struct ConwayPolynomial<859, 8> { using ZPZ = aerobus::zpz<859>; using type =
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<446>, ZPZV<672>, ZPZV<2»; }; //
        template<> struct ConwayPolynomial<859, 9> { using ZPZ = aerobus::zpz<859>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<648>, ZPZV<845>, ZPZV<857»;</pre>
04924
         }; // NOLINT
04925
              template<> struct ConwayPolynomial<863, 1> { using ZPZ = aerobus::zpz<863>; using type =
        POLYV<ZPZV<1>, ZPZV<858»; }; // NOLINT
04926
               template<> struct ConwayPolynomial<863, 2> { using ZPZ = aerobus::zpz<863>; using type =
        POLYV<ZPZV<1>, ZPZV<862>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<863, 3> { using ZPZ = aerobus::zpz<863>; using type =
04927
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<858»; }; // NOLINT
               template<> struct ConwayPolynomial<863, 4> { using ZPZ = aerobus::zpz<863>; using type =
04928
        04929
               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
        template<> struct ConwayPolynomial</pr>
### template
### struct ConwayPolynomial
#### type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<330>, ZPZV<62>, ZPZV<300>, ZPZV<5»; }; // NOLINT
04930
04931
               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<0>, ZPZV<1>, ZPZV<858»; }; // NOLINT
04932
              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
         NOLINT
04933
              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<381>, ZPZV<45, ZPZV<858»; };
         // NOLINT
04934
              template<> struct ConwayPolynomial<877, 1> { using ZPZ = aerobus::zpz<877>; using type =
        POLYV<ZPZV<1>, ZPZV<875»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 2> { using ZPZ = aerobus::zpz<877>; using type =
04935
         POLYV<ZPZV<1>, ZPZV<873>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 3> { using ZPZ = aerobus::zpz<877>; using type =
04936
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<875»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 4> { using ZPZ = aerobus::zpz<877>; using type =
04937
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<604>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<877, 5> { using ZPZ = aerobus::zpz<877>; using type =
04938
        POLYY<ZPZY<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<875»; }; // NOLINT template<> struct ConwayPolynomial<877, 6> { using ZPZ = aerobus::zpz<877>; using type =
        POLYV<2PZV<1>, 2PZV<0>, ZPZV<0>, ZPZV<629>, ZPZV<400>, ZPZV<855>, ZPZV<2»; }; // NOLINT
               template<> struct ConwayPolynomial<877, 7> { using ZPZ = aerobus::zpz<877>; using type =
04940
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<875»; }; // NOLIN template<> struct ConwayPolynomial<877, 8> { using ZPZ = aerobus::zpz<877>; using type =
04941
         POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<767>, ZPZV<319>, ZPZV<347>, ZPZV<2»; }; //
         NOLINT
               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<770>, ZPZV<278>, ZPZV<875»;
         }; // NOLINT
04943
               template<> struct ConwayPolynomial<881, 1> { using ZPZ = aerobus::zpz<881>; using type =
        POLYV<ZPZV<1>, ZPZV<878»; }; // NOLINT
               template<> struct ConwayPolynomial<881, 2> { using ZPZ = aerobus::zpz<881>; using type =
04944
        POLYV<ZPZV<1>, ZPZV<869>, ZPZV<3»; }; // NOLINT
               template<> struct ConwayPolynomial<881, 3> { using ZPZ = aerobus::zpz<881>; using type =
04945
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<878»; }; // NOLINT template<> struct ConwayPolynomial<881, 4> { using ZPZ = aerobus::zpz<881>; using type =
04946
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<447>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 5> { using ZPZ = aerobus::zpz<881>; using type =
04947
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<878»; }; // NOLINT
               template<> struct ConwayPolynomial<881, 6> { using ZPZ = aerobus::zpz<881>; using type =
04948
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<218>, ZPZV<419>, ZPZV<231>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<881, 7> { using ZPZ = aerobus::zpz<881>; using type =
04949
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<878»; }; // NOLINT
              template<> struct ConwayPolynomial<881, 8> { using ZPZ = aerobus::zpz<881>; using type
04950
         POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<21>, ZPZV<635>, ZPZV<490>, ZPZV<561>, ZPZV<56; //
04951
              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
04952
              template<> struct ConwayPolynomial<883, 1> { using ZPZ = aerobus::zpz<883>; using type =
```

9.3 aerobus.h

```
POLYV<ZPZV<1>, ZPZV<881»; };
              template<> struct ConwayPolynomial<883, 2> { using ZPZ = aerobus::zpz<883>; using type =
        POLYV<ZPZV<1>, ZPZV<879>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<883, 3> { using ZPZ = aerobus::zpz<883>; using type =
04954
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<6>, ZPZV<881»; }; // NOLINT template<> struct ConwayPolynomial<883, 4> { using ZPZ = aerobus::zpz<883>; using type =
04955
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<715>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<883, 5> { using ZPZ = aerobus::zpz<883>; using type =
04956
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<14>, ZPZV<881»; }; // NOLINT
04957
             template<> struct ConwayPolynomial<883, 6> { using ZPZ = aerobus::zpz<883>; using type =
        04958
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>, ZPZV<881»; };
              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<20»; }; //
             template<> struct ConwayPolynomial<883, 9> { using ZPZ = aerobus::zpz<883>; using type =
04960
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<360>, ZPZV<360>, ZPZV<557>, ZPZV<881»;
        }; // NOLINT
              template<> struct ConwayPolynomial<887, 1> { using ZPZ = aerobus::zpz<887>; using type =
        POLYV<ZPZV<1>, ZPZV<882»; }; // NOLINT
              template<> struct ConwayPolynomial<887, 2> { using ZPZ = aerobus::zpz<887>; using type =
04962
        POLYV<ZPZV<1>, ZPZV<885>, ZPZV<5»; }; // NOLINT
              template<> struct ConwayPolynomial<887, 3> { using ZPZ = aerobus::zpz<887>; using type =
04963
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<882»; }; // NOLINT template<> struct ConwayPolynomial<887, 4> { using ZPZ = aerobus::zpz<887>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<883>, ZPZV<5»; }; // NOLINT
04965
              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
04966
             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<28>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<887, 7> { using ZPZ = aerobus::zpz<887>; using type
04967
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<8>, ZPZV<8 , ZPZV<8
04968
             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
        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»;
04969
        }; // NOLINT
              template<> struct ConwayPolynomial<907, 1> { using ZPZ = aerobus::zpz<907>; using type =
04970
        POLYV<ZPZV<1>, ZPZV<905»; }; // NOLINT
              template<> struct ConwayPolynomial<907, 2> { using ZPZ = aerobus::zpz<907>; using type =
04971
        POLYV<ZPZV<1>, ZPZV<903>, ZPZV<2»: }: // NOLINT
04972
              template<> struct ConwayPolynomial<907, 3> { using ZPZ = aerobus::zpz<907>; using type =
        POLYY<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<905»; }; // NOLINT template<> struct ConwayPolynomial<907, 4> { using ZPZ = aerobus::zpz<907>; using type =
04973
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<14>, ZPZV<478>, ZPZV<2»; }; // NOLINT
             template<> struct ConwayPolynomial<907, 5> { using ZPZ = aerobus::zpz<907>; using type =
04974
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<905»; }; // NOLINT
              template<> struct ConwayPolynomial<907, 6> { using ZPZ = aerobus::zpz<907>; using type =
04975
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<626>, ZPZV<752>, ZPZV<266>, ZPZV<2»; }; // NOLINT
              template<> struct ConwayPolynomial<907, 7> { using ZPZ = aerobus::zpz<907>; using type =
04976
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<2>, ZPZV<2>, ZPZV<905»; }; // NOLINT
04977
              template<> struct ConwayPolynomial<907, 8> { using ZPZ = aerobus::zpz<907>; using type =
        POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<584>, ZPZV<518>, ZPZV<811>, ZPZV<2»; }; //
        NOLINT
              template<> struct ConwayPolynomial<907, 9> { using ZPZ = aerobus::zpz<907>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<783>, ZPZV<57>, ZPZV<905»;
        }; // NOLINT
04979
              template<> struct ConwayPolynomial<911, 1> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<894»; }; // NOLINT
              template<> struct ConwayPolynomial<911, 2> { using ZPZ = aerobus::zpz<911>; using type =
04980
        POLYV<ZPZV<1>, ZPZV<909>, ZPZV<17»; }; // NOLINT
              template<> struct ConwayPolynomial<911, 3> { using ZPZ = aerobus::zpz<911>; using type = VV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; }; // NOLINT
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<894»; };
             template<> struct ConwayPolynomial<911, 4> { using ZPZ = aerobus::zpz<911>; using type =
04982
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<11>, ZPZV<887>, ZPZV<17»; }; // NOLINT template<> struct ConwayPolynomial<911, 5> { using ZPZ = aerobus::zpz<911>; using type =
04983
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<894»; }; // NOLINT
04984
              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
04985
             template<> struct ConwayPolynomial<911, 7> { using ZPZ = aerobus::zpz<911>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<4>, ZPZV<4>, ZPZV<894»; }; // NOLINT template<> struct ConwayPolynomial<911, 8> { using ZPZ = aerobus::zpz<911>; using type =
04986
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<708>, ZPZV<590>, ZPZV<168>, ZPZV<17»; }; //
04987
             template<> struct ConwayPolynomial<911, 9> { using ZPZ = aerobus::zpz<911>; using type
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<679>, ZPZV<116>, ZPZV<894»;
        }; // NOLINT
04988
              template<> struct ConwayPolynomial<919, 1> { using ZPZ = aerobus::zpz<919>; using type =
        POLYV<ZPZV<1>, ZPZV<912»; }; // NOLINT
              template<> struct ConwayPolynomial<919, 2> { using ZPZ = aerobus::zpz<919>; using type =
        POLYV<ZPZV<1>, ZPZV<910>, ZPZV<7»; }; // NOLINT
04990
             template<> struct ConwayPolynomial<919, 3> { using ZPZ = aerobus::zpz<919>; using type =
        POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<912»; }; // NOLINT template<> struct ConwayPolynomial<919, 4> { using ZPZ = aerobus::zpz<919>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<602>, ZPZV<7»; }; // NOLINT
04991
```

172 File Documentation

```
04992
                   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
04993
                  template<> struct ConwayPolynomial<919, 6> { using ZPZ = aerobus::zpz<919>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<312>, ZPZV<817>, ZPZV<113>, ZPZV<7»; }; // NOLINT template<> struct ConwayPolynomial<919, 7> { using ZPZ = aerobus::zpz<919>; using type
04994
           POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0
                                                                                                                                                                          // NOLINT
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<708>, ZPZV<202>, ZPZV<504>, ZPZV<504>, ZPZV<7»; }; //
           template<> struct ConwayPolynomial<919, 9> { using ZPZ = aerobus::zpz<919>; using type =
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<410>, ZPZV<623>, ZPZV<912»;</pre>
04996
           }; // NOLINT
04997
                   template<> struct ConwayPolynomial<929, 1> { using ZPZ = aerobus::zpz<929>; using type =
           POLYV<ZPZV<1>, ZPZV<926»; }; // NOLINT
04998
                   template<> struct ConwayPolynomial<929, 2> { using ZPZ = aerobus::zpz<929>; using type =
           POLYV<ZPZV<1>, ZPZV<917>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<929, 3> { using ZPZ = aerobus::zpz<929>; using type =
04999
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<926»; }; // NOLINT
                   template<> struct ConwayPolynomial<929, 4> { using ZPZ = aerobus::zpz<929>; using type =
05000
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<787>, ZPZV<3»; }; // NOLINT
                   template<> struct ConwayPolynomial<929, 5> { using ZPZ = aerobus::zpz<929>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<926»; }; // NOLINT
                  template<> struct ConwayPolynomial<929, 6> { using ZPZ = aerobus::zpz<929>; using type =
05002
           POLYY<ZPZV<1>, ZPZV<2>, ZPZV<2>, ZPZV<805>, ZPZV<805>, ZPZV<86>, ZPZV<3»; ); // NOLINT template<> struct ConwayPolynomial<929, 7> { using ZPZ = aerobus::zpz<929>; using type
05003
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<926»; }; //
05004
                  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
05005
                  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<0>, ZPZV<481>, ZPZV<199>, ZPZV<926»;
           }; // NOLINT
                  template<> struct ConwayPolynomial<937, 1> { using ZPZ = aerobus::zpz<937>; using type =
05006
           POLYV<ZPZV<1>, ZPZV<932»; }; // NOLINT
05007
                   template<> struct ConwayPolynomial<937, 2> { using ZPZ = aerobus::zpz<937>; using type =
           POLYV<ZPZV<1>, ZPZV<934>, ZPZV<5»; }; // NOLINT
           template<> struct ConwayPolynomial<937, 3> { using ZPZ = aerobus::zpz<937>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<932»; }; // NOLINT
05008
05009
                   template<> struct ConwayPolynomial<937, 4> { using ZPZ = aerobus::zpz<937>; using type =
           POLYY<ZPZY<1>, ZPZY<0>, ZPZY<23>, ZPZY<585>, ZPZY<5»; }; // NOLINT template<> struct ConwayPolynomial<937, 5> { using ZPZ = aerobus::zpz<937>; using type =
05010
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<932»; }; // NOLINT 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
0.5011
                   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<0>, ZPZV<24>, ZPZV<932»; };
05013
                  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
                   template<> struct ConwayPolynomial<937, 9> { using ZPZ = aerobus::zpz<937>; using type =
05014
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<28>, ZPZV<533>, ZPZV<483>, ZPZV<932»;
           }; // NOLINT
05015
                   template<> struct ConwayPolynomial<941, 1> { using ZPZ = aerobus::zpz<941>; using type =
           POLYY<ZPZY<1>, ZPZY<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 2> { using ZPZ = aerobus::zpz<941>; using type =
05016
           POLYV<ZPZV<1>, ZPZV<940>, ZPZV<2»; }; // NOLINT
                   template<> struct ConwayPolynomial<941, 3> { using ZPZ = aerobus::zpz<941>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 4> { using ZPZ = aerobus::zpz<941>; using type =
05018
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<505>, ZPZV<2»; }; // NOLINT
                  template<> struct ConwayPolynomial<941, 5> { using ZPZ = aerobus::zpz<941>; using type =
05019
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<939»; }; // NOLINT
05020
                   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
05021
                  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<0>, ZPZV<4>, ZPZV<4>, ZPZV<4>, ZPZV<939»; }; // NOLINT template<> struct ConwayPolynomial<941, 8> { using ZPZ = aerobus::zpz<941>; using type =
05022
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<675>, ZPZV<590>, ZPZV
           NOLINT
                   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
05024
                   template<> struct ConwayPolynomial<947, 1> { using ZPZ = aerobus::zpz<947>; using type =
           POLYV<ZPZV<1>, ZPZV<945»; }; // NOLINT
                   template<> struct ConwayPolynomial<947, 2> { using ZPZ = aerobus::zpz<947>; using type =
05025
           POLYV<ZPZV<1>, ZPZV<943>, ZPZV<2»; }; // NOLINT
                   template<> struct ConwayPolynomial<947, 3> { using ZPZ = aerobus::zpz<947>; using type =
05026
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<945»; }; // NOLINT template<> struct ConwayPolynomial<947, 4> { using ZPZ = aerobus::zpz<947>; using type =
05027
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<8>, ZPZV<894>, ZPZV<2»; }; // NOLINT template<> struct ConwayPolynomial<947, 5> { using ZPZ = aerobus::zpz<947>; using type =
05028
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<945»; }; // NOLINT
                   template<> struct ConwayPolynomial<947, 6> { using ZPZ = aerobus::zpz<947>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<880>, ZPZV<787>, ZPZV<95>, ZPZV<2»; }; // NOLINT
          template<> struct ConwayPolynomial<947, 7> { using ZPZ = aerobus::zpz<947>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<6>, ZPZV<6>, ZPZV<6>; // NOLINT template<> struct ConwayPolynomial<947, 8> { using ZPZ = aerobus::zpz<947>; using type =
05030
05031
```

9.3 aerobus.h 173

```
POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<845>, ZPZV<597>, ZPZV<581>, ZPZV<2*; }; //
05032
                 template<> struct ConwayPolynomial<947, 9> { using ZPZ = aerobus::zpz<947>; using type =
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<26>, ZPZV<269>, ZPZV<269>, ZPZV<808>, ZPZV<945»;
           }; // NOLINT
05033
                  template<> struct ConwavPolynomial<953, 1> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<950»; }; // NOLINT
                  template<> struct ConwayPolynomial<953, 2> { using ZPZ = aerobus::zpz<953>; using type =
05034
           POLYV<ZPZV<1>, ZPZV<947>, ZPZV<3»; }; // NOLINT
05035
                  template<> struct ConwayPolynomial<953, 3> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<7>, ZPZV<950»; }; // NOLINT template<> struct ConwayPolynomial<953, 4> { using ZPZ = aerobus::zpz<953>; using type =
05036
          POLYY<ZPZY<1>, ZPZV<0>, ZPZV<1>, ZPZV<65>, ZPZV<685>, ZPZV<6»; }; // NOLINT 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
05038
                  template<> struct ConwayPolynomial<953, 6> { using ZPZ = aerobus::zpz<953>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<507, ZPZV<829>, ZPZV<730>, ZPZV<3»; }; // NOLINT template<> struct ConwayPolynomial<953, 7> { using ZPZ = aerobus::zpz<953>; using type
05039
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<5>, ZPZV<5>, ZPZV<5>, ZPZV<50»; }; // NOLINT
                  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<108>, ZPZV<3»; }; //
           NOLINT
05041
                 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
                   template<> struct ConwayPolynomial<967, 1> { using ZPZ = aerobus::zpz<967>; using type =
           POLYV<ZPZV<1>, ZPZV<962»; }; // NOLINT
                  template<> struct ConwayPolynomial<967, 2> { using ZPZ = aerobus::zpz<967>; using type =
05043
          POLYV<ZPZV<1>, ZPZV<965>, ZPZV<5»; }; // NOLINT
                 template<> struct ConwayPolynomial<967, 3> { using ZPZ = aerobus::zpz<967>; using type =
05044
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 4> { using ZPZ = aerobus::zpz<967>; using type =
05045
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<3>, ZPZV<963>, ZPZV<5»; }; // NOLINT
05046
                 template<> struct ConwayPolynomial<967, 5> { using ZPZ = aerobus::zpz<967>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<2>, ZPZV<962»; }; // NOLINT template<> struct ConwayPolynomial<967, 6> { using ZPZ = aerobus::zpz<967>; using type =
05047
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<831>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<967, 7> { using ZPZ = aerobus::zpz<967>; using type
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<962»; };
                 template<> struct ConwayPolynomial<967, 8> { using ZPZ = aerobus::zpz<967>; using type =
05049
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<840>, ZPZV<502>, ZPZV<136>, ZPZV<5»; }; //
           NOLINT
                  template<> struct ConwayPolynomial<967, 9> { using ZPZ = aerobus::zpz<967>; using type =
05050
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<51>, ZPZV<512>, ZPZV<783>, ZPZV<962»;
05051
                  template<> struct ConwayPolynomial<971, 1> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<965»; }; // NOLINT
                 template<> struct ConwayPolynomial<971, 2> { using ZPZ = aerobus::zpz<971>; using type =
05052
          POLYV<ZPZV<1>, ZPZV<970>, ZPZV<6»; }; // NOLINT
                  template<> struct ConwayPolynomial<971, 3> { using ZPZ = aerobus::zpz<971>; using type =
05053
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<965»; }; // NOLINT template<> struct ConwayPolynomial<971, 4> { using ZPZ = aerobus::zpz<971>; using type =
          05055
                  template<> struct ConwayPolynomial<971, 5> { using ZPZ = aerobus::zpz<971>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<40, ZPZV<965»; }; // NOLINT
                  template<> struct ConwayPolynomial<971, 6> { using ZPZ = aerobus::zpz<971>; using type =
05056
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<970>, ZPZV<729>, ZPZV<718>, ZPZV<6»; }; // NOLINT
                 template<> struct ConwayPolynomial<971,
                                                                                           7> { using ZPZ = aerobus::zpz<971>; using type
05057
          POLYY<ZPZY<1>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<0>, ZPZY<13-, ZPZY<958*; }/ NOLI template<> struct ConwayPolynomial<971, 8> { using ZPZ = aerobus::zpz<971>; using type =
05058
           POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<281>, ZPZV<206>, ZPZV<6»; }; //
           NOLINT
05059
                  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<0>, ZPZV<0>, ZPZV<0>, ZPZV<805>, ZPZV<473>, ZPZV<965»;
           }; // NOLINT
05060
                 template<> struct ConwayPolynomial<977, 1> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<974»; }; // NOLINT
                  template<> struct ConwayPolynomial<977, 2> { using ZPZ = aerobus::zpz<977>; using type =
05061
          POLYV<ZPZV<1>, ZPZV<972>, ZPZV<3»; }; // NOLINT
                  template<> struct ConwayPolynomial 977, 3> { using ZPZ = aerobus::zpz<977>; using type =
05062
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<974»; }; // NOLINT template<> struct ConwayPolynomial<977, 4> { using ZPZ = aerobus::zpz<977>; using type =
05063
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<600>, ZPZV<500>; ZPZV<500>, ZPZV<500>; ZPZV<500>; ZPZV<500>; ZPZV<500>; ZPZV<500>; ZPZV<500; ZPZV<
05064
          POLYY<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0), ZPZV<11>, ZPZV<974»; }; // NOLINT template<> struct ConwayPolynomial<977, 6> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<2PZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<729>, ZPZV<830>, ZPZV<753>, ZPZV<3»; }; // NOLINT
05066
                 template<> struct ConwayPolynomial<977, 7> { using ZPZ = aerobus::zpz<977>; using type =
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<7>, ZPZV<7>, ZPZV<974%; }; // NOLINT template<> struct ConwayPolynomial<977, 8> { using ZPZ = aerobus::zpz<977>; using type =
05067
          POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<855>, ZPZV<807>, ZPZV<77>, ZPZV<3»; };
          template<> struct ConwayPolynomial<977, 9> { using ZPZ = aerobus::zpz<977>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<450>, ZPZV<450>, ZPZV<740>, ZPZV<
           }; // NOLINT
05069
                  template<> struct ConwayPolynomial<983, 1> { using ZPZ = aerobus::zpz<983>; using type =
           POLYV<ZPZV<1>, ZPZV<978»; }; // NOLINT
```

174 File Documentation

```
template<> struct ConwayPolynomial<983, 2> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<981>, ZPZV<5»; }; // NOLINT
05071
           template<> struct ConwayPolynomial<983, 3> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<1>, ZPZV<978»; }; // NOLINT template<> struct ConwayPolynomial<983, 4> { using ZPZ = aerobus::zpz<983>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<5>, ZPZV<567>, ZPZV<5»; }; // NOLINT
05072
           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
05074
           template<> struct ConwayPolynomial<983, 6> { using ZPZ = aerobus::zpz<983>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<849>, ZPZV<296>, ZPZV<228>, ZPZV<5»; }; // NOLINT template<> struct ConwayPolynomial<983, 7> { using ZPZ = aerobus::zpz<983>; using type =
05075
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<3>, ZPZV<978»; }; // NOLINT
           template<> struct ConwayPolynomial<983, 8> { using ZPZ = aerobus::zpz<983>; using type =
05076
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<738>, ZPZV<276>, ZPZV<530>, ZPZV<5»; }; //
       NOLINT
05077
           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
            template<> struct ConwayPolynomial<991, 1> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<985»; }; // NOLINT
            template<> struct ConwayPolynomial<991, 2> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<989>, ZPZV<6»; }; // NOLINT
05080
           template<> struct ConwayPolynomial<991, 3> { using ZPZ = aerobus::zpz<991>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<985»; }; // NOLINT template<> struct ConwayPolynomial<991, 4> { using ZPZ = aerobus::zpz<991>; using type =
05081
      POLYY<ZPZY<1>, ZPZV<0>, ZPZV<10>, ZPZV<794>, ZPZV<6»; }; // NOLINT template<> struct ConwayPolynomial<991, 5> { using ZPZ = aerobus::zpz<991>; using type =
05082
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<985»; }; // NOLINT
05083
           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 template<> struct ConwayPolynomial<991, 7> { using ZPZ = aerobus::zpz<991>; using type
05084
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<7>, ZPZV<78, ZPZV<985»; };
           template<> struct ConwayPolynomial<991, 8> { using ZPZ = aerobus::zpz<991>; using type =
05085
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<15>, ZPZV<786>, ZPZV<234>, ZPZV<6*; }; //
       NOLINT
05086
           template<> struct ConwayPolynomial<991, 9> { using ZPZ = aerobus::zpz<991>; using type =
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<9>, ZPZV<9>, ZPZV<466>, ZPZV<222>, ZPZV<985»;
       }; // NOLINT
05087
            template<> struct ConwayPolynomial<997, 1> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<990»; }; // NOLINT
05088
           template<> struct ConwayPolynomial<997, 2> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<995>, ZPZV<7»; }; // NOLINT
template<> struct ConwayPolynomial<997, 3> { using ZPZ = aerobus::zpz<997>; using type =
05089
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<2>, ZPZV<990»; }; // NOLINT
           template<> struct ConwayPolynomial<997, 4> { using ZPZ = aerobus::zpz<997>; using type =
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<4>, ZPZV<622>, ZPZV<7»; }; // NOLINT
      template<> struct ConwayPolynomial<997, 5> { using ZPZ = aerobus::zpz<997>; using type =
POLYV<2PZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<10>, ZPZV<90>, ZPZV<90>; }; // NOLINT
05091
           template<> struct ConwayPolynomial<997, 6> { using ZPZ = aerobus::zpz<997>; using type =
05092
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<981>, ZPZV<58>, ZPZV<260>, ZPZV<7»; }; // NOLINT
05093
           template<> struct ConwayPolynomial<997,
                                                          7> { using ZPZ = aerobus::zpz<997>; using type
      POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<1>, ZPZV<990»; }; //
           template<> struct ConwayPolynomial<997, 8> { using ZPZ = aerobus::zpz<997>; using type =
05094
       POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<934>, ZPZV<473>, ZPZV<241>, ZPZV<241>, ZPZV<7»; }; //
      NOLINT
      template<> struct ConwayPolynomial<997, 9> { using ZPZ = aerobus::zpz<997>; using type = POLYV<ZPZV<1>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<0>, ZPZV<3>, ZPZV<732>, ZPZV<616>, ZPZV<990»;
05095
05096 #endif // DO_NOT_DOCUMENT
05097 } // namespace aerobus
05098 #endif // AEROBUS_CONWAY_IMPORTS
05099
05100 #endif // __INC_AEROBUS__ // NOLINT
```

Chapter 10

Examples

10.1 QuotientRing

inject a 'constant' in quotient ring

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

Template Parameters

x a 'constant' from Ring point of view

10.2 type_list

A list of types <int, double, float>

A list of types <int, double, float>

Template Parameters

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

10.3 i32::template

inject a native constant

inject a native constant

Template Parameters

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

10.4 i32::add_t

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

Template Parameters

| v1 | a value in i32 |
|----|----------------|
| v2 | a value in i32 |

10.5 i32::sub_t

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

Template Parameters

| v1 | a value in i32 |
|----|----------------|
| v2 | a value in i32 |

10.6 i32::mul_t

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

Template Parameters

| v1 | a value in i32 |
|----|----------------|
| v2 | a value in i32 |

10.7 i32::div_t

 $\label{eq:continuous} \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 2>> -> i32::val < 3> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 2>> -> i32::val < 3> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 2>> -> i32::val < 3> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 2>> -> i32::val < 3> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7>, i32::val < 7>, i32::val < 7> -> i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> -> i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> } \\ \mbox{division operator yields v1 / v2 < i32::val < 7> }$

| v1 | a value in i32 |
|----|----------------|
| v2 | a value in i32 |

10.11 i32::gcd_t 177

10.8 i32::gt_t

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

Template Parameters

| v1 | a value in i32 |
|----|----------------|
| v2 | a value in i32 |

10.9 i32::eq_t

$$\label{eq:constant} \begin{split} &\text{equality operator (type) yields v1 == v2 as std::integral_constant<bool> < i32::val<2>, i32::val<2>> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant<bool> < i32::val<2>, i32::val<2>> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant<bool> < i32::val<2>, i32::val<2>> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant<bool> < i32::val<2>, i32::val<2>> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std::integral_constant
 < i32::val<2> \\ &\text{equality operator (type) yields v1 == v2 as std:$$

Template Parameters

| v1 | a value in i32 |
|----|----------------|
| v2 | a value in i32 |

10.10 i32::eq_v

equality operator (boolean value)

equality operator (boolean value)

Template Parameters

| v1 | |
|----|--|
| v2 | <i32::val<1>, i32::val<1>></i32::val<1> |

10.11 i32::gcd_t

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

| v1 | a value in i32 |
|----|----------------|
| v2 | a value in i32 |

10.12 i32::pos_t

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

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

Template Parameters

v a value in i32

10.13 i32::pos_v

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

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

Template Parameters

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

10.14 i64::template

injects constant as an i64 value

injects constant as an i64 value

Template Parameters

x inject_constant_t<2>

10.15 i64::add_t

addition operator

addition operator

| v1 | : an element of aerobus::i64::val |
|----|---|
| v2 | : an element of aerobus::i64::val <i64::val <1="">, i64::val <2>></i64::val> |

10.19 i64::mod_t 179

10.16 i64::sub_t

substraction operator

substraction operator

Template Parameters

| v1 | : an element of aerobus::i64::val |
|----|---|
| v2 | : an element of aerobus::i64::val <i64::val <1="">, i64::val <2>></i64::val> |

10.17 i64::mul_t

multiplication operator

multiplication operator

Template Parameters

| v1 | : an element of aerobus::i64::val |
|----|---|
| v2 | : an element of aerobus::i64::val <i64::val <1="">, i64::val <2>></i64::val> |

10.18 i64::div_t

division operator integer division

division operator integer division

Template Parameters

| v1 | : an element of aerobus::i64::val |
|----|---|
| v2 | : an element of aerobus::i64::val <i64::val <1="">, i64::val <2>></i64::val> |

10.19 i64::mod_t

modulus operator

modulus operator

| v1 | : an element of aerobus::i64::val | |
|----|--|--|
| v2 | : an element of aerobus::i64::val <i64::val <6="">, i64::val <15>></i64::val> | |

10.20 i64::gt t

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

Template Parameters

| v1 | : an element of aerobus::i64::val | |
|----|---|--|
| v2 | : an element of aerobus::i64::val <i64::val <2="">, i64::val <1>></i64::val> | |

10.21 i64::lt_t

Template Parameters

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

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

10.22 i64::lt_v

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

Template Parameters

| v1 | : an element of aerobus::i64::val | |
|----|---|--|
| v2 | : an element of aerobus::i64::val <i64::val <1="">, i64::val <2>></i64::val> | |

10.23 i64::eq_t

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

| v1 | : an element of aerobus::i64::val | |
|----|---|--|
| v2 | : an element of aerobus::i64::val <i64::val <2="">, i64::val <2>></i64::val> | |

10.27 i64::pos_v 181

10.24 i64::eq_v

equality operator yields v1 == v2 as boolean value

equality operator yields v1 == v2 as boolean value

Template Parameters

| v1 | : an element of aerobus::i64::val | |
|----|---|--|
| v2 | : an element of aerobus::i64::val <i64::val <2="">, i64::val <2>></i64::val> | |

10.25 i64::gcd_t

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

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

Template Parameters

| v1 | : an element of aerobus::i64::val | |
|----|--|--|
| v2 | : an element of aerobus::i64::val <i64::val <6="">, i64::val <15>></i64::val> | |

10.26 i64::pos_t

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

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

Template Parameters

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

10.27 i64::pos_v

positivity yields v > 0 as boolean value

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

Template Parameters

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

10.28 polynomial

makes the constant (native type) polynomial a_0

makes the constant (native type) polynomial a_0

Template Parameters

x <i32>::template inject_constant_t<2>

10.29 q32::add_t

addition operator

addition operator

Template Parameters

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

10.30 FractionField

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

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

Template Parameters

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

10.31 aerobus::ContinuedFraction

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

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

| values | are |
|--------|---------|
| | int64_t |

10.32 Pl_fraction::val

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

10.32 Pl_fraction::val

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

10.33 E_fraction::val

approximation of e -> 2.718...

approximation of $e \rightarrow 2.718...$

Index

```
abs t
                                                               sinh. 28
                                                               SQRT2_fraction, 28
     aerobus, 21
add t
                                                               SQRT3 fraction, 28
     aerobus::i32, 48
                                                               stirling_signed_t, 28
     aerobus::i64, 51
                                                               stirling_unsigned_t, 29
     aerobus::polynomial < Ring >, 57
                                                               tan, 29
     aerobus::Quotient < Ring, X >, 64
                                                               tanh, 29
     aerobus::zpz, 85
                                                               taylor, 29
addfractions_t
                                                               vadd_t, 30
     aerobus, 21
                                                               vmul t, 30
aerobus, 17
                                                          aerobus::ContinuedFraction < a0 >, 44
     abs t, 21
                                                               type, 45
     addfractions_t, 21
                                                               val, 45
     aligned_malloc, 30
                                                          aerobus::ContinuedFraction < a0, rest... >, 45
     alternate t, 21
                                                               type, 46
                                                               val. 46
     alternate_v, 31
                                                          aerobus::ContinuedFraction< values >, 44
     asin, 21
     asinh, 22
                                                          aerobus::ConwayPolynomial, 46
     atan, 22
                                                          aerobus::i32, 46
     atanh, 22
                                                               add_t, 48
    bernoulli t, 22
                                                               div_t, 48
    bernoulli v, 31
                                                               eq t, 48
    combination t, 23
                                                               eq v, 49
    combination_v, 31
                                                               gcd_t, 48
    cos, 23
                                                               gt_t, 48
     cosh, 23
                                                               inject constant t, 48
     E_fraction, 23
                                                               inject_ring_t, 48
    exp, 24
                                                               inner_type, 48
     expm1, 24
                                                               is_euclidean_domain, 49
     factorial t, 24
                                                               is field, 49
     factorial v, 32
                                                               It t, 48
    field, 31
                                                               mod_t, 48
     fpq32, 24
                                                               mul t, 49
     fpq64, 24
                                                               one, 49
     FractionField, 25
                                                               pos_t, 49
    gcd_t, 25
                                                               pos_v, 50
     geometric_sum, 25
                                                               sub_t, 49
    Inp1, 25
                                                               zero, 49
     make_q32_t, 26
                                                          aerobus::i32::val< x >, 72
     make_q64_t, 26
                                                               enclosing_type, 73
     makefraction_t, 26
                                                               eval, 73
     mulfractions t, 26
                                                               get, 73
     pi64, 27
                                                               is_zero_t, 73
     PI fraction, 27
                                                               to_string, 74
     pow t, 27
                                                               v, 74
    pq64, 27
                                                          aerobus::i64, 50
    q32, 27
                                                               add_t, 51
     q64, 27
                                                               div_t, 51
                                                               eq_t, 51
     sin, 28
```

| eq_v, 53 | lt_t, 58 |
|----------------------------------|--|
| gcd_t, 51 | mod_t, 59 |
| gt_t, 52 | monomial_t, 59 |
| gt_v, 53 | mul_t, 59 |
| inject_constant_t, 52 | one, 60 |
| inject_ring_t, 52 | pos_t, 60 |
| inner_type, 52 | pos_v, 61 |
| is_euclidean_domain, 54 | simplify_t, 60 |
| is_field, 54 | sub_t, 60 |
| lt_t, 52 | X, 61 |
| lt_v, 54 | zero, 61 |
| mod_t, 52 | aerobus::polynomial< Ring >::val< coeffN >, 81 |
| mul_t, 52 | aN, 82 |
| one, 53 | coeff_at_t, 82 |
| pos_t, 53 | degree, 84 |
| pos_v, 54 | enclosing_type, 83 |
| sub_t, 53 | eval, 83 |
| zero, <u>53</u> | is_zero_t, 83 |
| aerobus::i64::val< x >, 74 | is_zero_v, 84 |
| enclosing_type, 75 | strip, 83 |
| eval, 75 | to_string, 83 |
| get, 75 | aerobus::polynomial < Ring >::val < coeffN >::coeff_at < |
| is_zero_t, 75 | index, $E >$, 43 |
| to_string, 76 | aerobus::polynomial< Ring >::val< coeffN >::coeff_at< |
| v, 76 | index, std::enable_if_t<(index< 0 index > |
| aerobus::internal, 32 | 0)>>, 43 |
| index_sequence_reverse, 36 | type, 43 |
| is_instantiation_of_v, 36 | aerobus::polynomial< Ring >::val< coeffN >::coeff_at< |
| make_index_sequence_reverse, 35 | index, std::enable_if_t<(index==0)>>, 44 |
| type_at_t, 35 | type, 44 |
| aerobus::is_prime< n >, 54 | aerobus::polynomial< Ring >::val< coeffN, coeffs >, |
| value, 55 | 76 |
| aerobus::IsEuclideanDomain, 41 | aN, 77 |
| aerobus::IsField, 41 | coeff_at_t, 77 |
| aerobus::IsRing, 42 | degree, 79 |
| aerobus::known_polynomials, 36 | enclosing_type, 77 |
| bernoulli, 37 | eval, 78 |
| | |
| bernstein, 37 | is_zero_t, 78 |
| chebyshev_T, 37 | is_zero_v, 79 |
| chebyshev_U, 38 | strip, 78 |
| hermite_kind, 40 | to_string, 78 aerobus::Quotient< Ring, X >, 62 |
| hermite_phys, 38 | • |
| hermite_prob, 38 | add_t, 64 |
| laguerre, 39 | div_t, 64 |
| legendre, 39 | eq_t, 64 |
| physicist, 40 | eq_v, 66 |
| probabilist, 40 | inject_constant_t, 64 |
| aerobus::polynomial < Ring >, 55 | inject_ring_t, 65 |
| add_t, 57 | is_euclidean_domain, 66 |
| derive_t, 57 | mod_t, 65 |
| div_t, 57 | mul_t, 65 |
| eq_t, 57 | one, 65 |
| gcd_t, 58 | pos_t, 65 |
| gt_t, 58 | pos_v, 66 |
| inject_constant_t, 58 | zero, 66 |
| inject_ring_t, 58 | aerobus::Quotient< Ring, X >::val< V >, 79 |
| is_euclidean_domain, 61 | type, 80 |
| is_field, 61 | aerobus::type_list< Ts >, 68 |
| | |

| at, 69 | aerobus, 21 |
|--|--|
| concat, 69 | asinh |
| insert, 69 | aerobus, 22 |
| length, 70 | at |
| push_back, 69 | aerobus::type_list< Ts >, 69 |
| push_front, 70 | atan |
| remove, 70 | aerobus, 22 |
| aerobus::type_list< Ts >::pop_front, 61 | atanh |
| tail, 62 | aerobus, 22 |
| type, 62 | dolobdo, EE |
| aerobus::type_list< Ts >::split< index >, 67 | bernoulli |
| head, 67 | aerobus::known_polynomials, 37 |
| tail, 67 | bernoulli_t |
| | aerobus, 22 |
| aerobus::type_list<>>, 70 | bernoulli_v |
| concat, 71 | |
| insert, 71 | aerobus, 31 |
| length, 72 | bernstein |
| push_back, 71 | aerobus::known_polynomials, 37 |
| push_front, 71 | chebyshev_T |
| aerobus::zpz, 84 | aerobus::known_polynomials, 37 |
| add_t, 85 | chebyshev U |
| div_t, 86 | • – |
| eq_t, 86 | aerobus::known_polynomials, 38 |
| eq_v, 89 | coeff_at_t |
| gcd_t, 86 | aerobus::polynomial < Ring >::val < coeffN >, 82 |
| gt_t, 86 | aerobus::polynomial< Ring >::val< coeffN, coeffs |
| gt_v, 89 | >, 77 |
| inject_constant_t, 87 | combination_t |
| inner_type, 87 | aerobus, 23 |
| is_euclidean_domain, 89 | combination_v |
| is_field, 89 | aerobus, 31 |
| lt_t, 87 | concat |
| lt_v, 89 | aerobus::type_list< Ts >, 69 |
| mod_t, 87 | aerobus::type_list<>, 71 |
| mul_t, 88 | cos |
| one, 88 | aerobus, 23 |
| pos_t, 88 | cosh |
| pos_v, 90 | aerobus, 23 |
| sub_t, 88 | dance |
| zero, 89 | degree |
| aerobus:: $zpz ::val < x >$, 80 | aerobus::polynomial < Ring >::val < coeffN >, 84 |
| enclosing_type, 80 | aerobus::polynomial< Ring >::val< coeffN, coeffs |
| eval, 81 | >, 79 |
| get, 81 | derive_t |
| is_zero_t, 80 | aerobus::polynomial< Ring >, 57 |
| to_string, 81 | div_t |
| v, 81 | aerobus::i32, 48 |
| aligned_malloc | aerobus::i64, 51 |
| aerobus, 30 | aerobus::polynomial< Ring >, 57 |
| alternate_t | aerobus::Quotient< Ring, X >, 64 |
| aerobus, 21 | aerobus::zpz, 86 |
| alternate_v | |
| aerobus, 31 | E_fraction |
| aN | aerobus, 23 |
| aerobus::polynomial< Ring >::val< coeffN >, 82 | enclosing_type |
| aerobus::polynomial< Ring >::val< coeffN, coeffs | aerobus::i32::val< x >, 73 |
| >, 77 | aerobus::i64::val< x >, 75 |
| asin | aerobus::polynomial< Ring >::val< coeffN >, 83 |

| aerobus::polynomial< Ring >::val< coeffN, coeffs >, 77 | head aerobus::type_list< Ts >::split< index >, 67 |
|--|---|
| aerobus::zpz $<$ p $>$::val $<$ x $>$, 80 | hermite_kind |
| eq_t aerobus::i32, 48 | aerobus::known_polynomials, 40 hermite_phys |
| aerobus::i64, 51 | aerobus::known polynomials, 38 |
| aerobus::polynomial < Ring >, 57 | hermite_prob |
| aerobus::Quotient< Ring, X >, 64 | aerobus::known_polynomials, 38 |
| aerobus::zpz, 86 | |
| eq_v | index_sequence_reverse |
| aerobus::i32, 49 | aerobus::internal, 36 |
| aerobus::i64, 53 | inject_constant_t |
| aerobus::Quotient< Ring, X >, 66 | aerobus::i32, 48 |
| aerobus::zpz, 89 | aerobus::i64, 52 |
| eval | aerobus::polynomial < Ring >, 58 |
| aerobus::i32::val $<$ x $>$, 73 | aerobus::Quotient< Ring, X >, 64 |
| aerobus::i64::val $< x >$, 75 | aerobus::zpz, 87 |
| aerobus::polynomial< Ring >::val< coeffN >, 83 | inject_ring_t |
| aerobus::polynomial< Ring >::val< coeffN, coeffs | aerobus::i32, 48 aerobus::i64, 52 |
| >, 78 | aerobus::io4, 52 aerobus::polynomial< Ring >, 58 |
| aerobus::zpz $<$ p $>$::val $<$ x $>$, 81 | aerobus::Quotient< Ring, X >, 65 |
| exp | inner_type |
| aerobus, 24 | aerobus::i32, 48 |
| expm1 | aerobus::i64, 52 |
| aerobus, 24 | aerobus::zpz, 87 |
| factorial_t | insert |
| aerobus, 24 | aerobus::type_list< Ts >, 69 |
| factorial_v | aerobus::type_list<>,71 |
| aerobus, 32 | Introduction, 1 |
| field | is_euclidean_domain |
| aerobus, 31 | aerobus::i32, 49 |
| fpq32 | aerobus::i64, 54 |
| aerobus, 24 | aerobus::polynomial< Ring >, 61 |
| fpq64 | aerobus::Quotient< Ring, X >, 66 |
| aerobus, 24 | aerobus::zpz, 89 |
| FractionField | is_field |
| aerobus, 25 | aerobus::i32, 49 |
| | aerobus::i64, 54 |
| gcd_t | aerobus::polynomial < Ring >, 61 |
| aerobus, 25 | aerobus::zpz, 89 |
| aerobus::i32, 48 | is_instantiation_of_v |
| aerobus::i64, 51 | aerobus::internal, 36 |
| aerobus::polynomial< Ring >, 58 | is_zero_t |
| aerobus::zpz, 86 | aerobus::i32::val $< x >$, 73 |
| geometric_sum | aerobus::i64::val $< x >$, 75 |
| aerobus, 25 | aerobus::polynomial< Ring >::val< coeffN >, 83 |
| get | aerobus::polynomial< Ring >::val< coeffN, coeffs |
| aerobus::i32::val $< x >$, 73 | >, 78 |
| aerobus::i64::val $< x >$, 75 | aerobus::zpz $<$ p $>$::val $<$ x $>$, 80 |
| aerobus::zpz::val< x >, 81 | is_zero_v |
| gt_t | aerobus::polynomial< Ring >::val< coeffN >, 84 |
| aerobus::i32, 48 aerobus::i64, 52 | aerobus::polynomial< Ring >::val< coeffN, coeffs |
| | >, 79 |
| aerobus::polynomial< Ring >, 58 aerobus::zpz, 86 | laguerre |
| | laguerre aerobus::known_polynomials, 39 |
| gt_v aerobus::i64, 53 | legendre |
| aerobus::zpz, 89 | aerobus::known_polynomials, 39 |
| 23.0000Epz \ p > , 00 | acrosaciiiomi_poiyriomiaio, 00 |

| length | aerobus::i32, 50 |
|--|--|
| aerobus::type_list< Ts >, 70 | aerobus::i64, 54 |
| aerobus::type_list<>, 72 | aerobus::polynomial < Ring >, 61 |
| Inp1 | aerobus::Quotient< Ring, X >, 66 |
| aerobus, 25 | aerobus::zpz $<$ p $>$, $\frac{90}{}$ |
| lt_t | pow_t |
| aerobus::i32, 48 | aerobus, 27 |
| aerobus::i64, <mark>52</mark> | pq64 |
| aerobus::polynomial $<$ Ring $>$, 58 | aerobus, 27 |
| aerobus:: $zpz $, 87 | probabilist |
| lt_v | aerobus::known_polynomials, 40 |
| aerobus::i64, 54 | push_back |
| aerobus::zpz $<$ p $>$, 89 | aerobus::type_list< Ts >, 69 |
| | aerobus::type_list<>, 71 |
| make_index_sequence_reverse | push_front |
| aerobus::internal, 35 | aerobus::type_list< Ts >, 70 |
| make_q32_t | aerobus::type_list<>, 71 |
| aerobus, 26 | , , , , , , , , , , , , , , , , , , , |
| make_q64_t | q32 |
| aerobus, 26 | aerobus, 27 |
| makefraction_t | q64 |
| aerobus, 26 | aerobus, 27 |
| mod_t | , |
| aerobus::i32, 48 | README.md, 91 |
| aerobus::i64, 52 | remove |
| aerobus::polynomial < Ring >, 59 | aerobus::type_list< Ts >, 70 |
| aerobus::Quotient< Ring, X >, 65 | 71 - 7 |
| aerobus:: $zpz $, 87 | simplify_t |
| monomial_t | aerobus::polynomial< Ring >, 60 |
| | sin |
| aerobus::polynomial< Ring >, 59 | aerobus, 28 |
| mul_t | sinh |
| aerobus::i32, 49 | aerobus, 28 |
| aerobus::i64, 52 | SQRT2_fraction |
| aerobus::polynomial < Ring >, 59 | aerobus, 28 |
| aerobus::Quotient< Ring, X >, 65 | SQRT3 fraction |
| aerobus::zpz, 88 | aerobus, 28 |
| mulfractions_t | src/aerobus.h, 91 |
| aerobus, 26 | stirling_signed_t |
| | - - |
| one | aerobus, 28 |
| aerobus::i32, 49 | stirling_unsigned_t |
| aerobus::i64, 53 | aerobus, 29 |
| aerobus::polynomial < Ring >, 60 | strip |
| aerobus::Quotient $<$ Ring, $X >$, 65 | aerobus::polynomial< Ring >::val< coeffN >, 83 |
| aerobus::zpz $<$ p $>$, 88 | aerobus::polynomial< Ring >::val< coeffN, coeffs |
| | >, 78 |
| physicist | sub_t |
| aerobus::known_polynomials, 40 | aerobus::i32, 49 |
| pi64 | aerobus::i64, 53 |
| aerobus, 27 | aerobus::polynomial $<$ Ring $>$, 60 |
| PI_fraction | aerobus::zpz, 88 |
| aerobus, 27 | |
| pos_t | tail |
| aerobus::i32, 49 | aerobus::type_list< Ts >::pop_front, 62 |
| | aerobus::type_list< Ts >::split< index >, 67 |
| aerobus::i64, 53 | are conserved by the first of t |
| aerobus::i64, 53 aerobus::polynomial < Ring >, 60 | tan |
| | — |
| aerobus::polynomial < Ring >, 60 | tan |
| aerobus::polynomial $<$ Ring $>$, 60 aerobus::Quotient $<$ Ring, $X >$, 65 | tan aerobus, 29 |

```
taylor
    aerobus, 29
to_string
    aerobus::i32::val< x >, 74
    aerobus::i64::val < x > , 76
    aerobus::polynomial< Ring >::val< coeffN >, 83
    aerobus::polynomial< Ring >::val< coeffN, coeffs
         >, 78
     aerobus::zpz ::val < x >, 81
type
    aerobus::ContinuedFraction < a0 >, 45
    aerobus::ContinuedFraction< a0, rest... >, 46
     aerobus::polynomial< Ring >::val< coeffN
         >::coeff_at< index, std::enable_if_t<(index<
         0 \mid | index > 0) > >, 43
     aerobus::polynomial< Ring
                                  >::val< coeffN
         >::coeff at<index, std::enable if t<(index==0)>
    aerobus::Quotient< Ring, X >::val< V >, 80
    aerobus::type_list< Ts >::pop_front, 62
type_at_t
     aerobus::internal, 35
     aerobus::i32::val< x >, 74
     aerobus::i64::val < x > , 76
     aerobus::zpz ::val < x >, 81
vadd t
     aerobus, 30
val
    aerobus::ContinuedFraction < a0 >, 45
    aerobus::ContinuedFraction< a0, rest... >, 46
value
     aerobus::is_prime< n >, 55
vmul t
    aerobus, 30
Χ
     aerobus::polynomial < Ring >, 61
zero
    aerobus::i32, 49
    aerobus::i64, 53
    aerobus::polynomial < Ring >, 61
    aerobus::Quotient < Ring, X >, 66
     aerobus::zpz, 89
```