🧩 Category 1 – Template Traits and Fundamentals

1. Implement your own is_same, is_integral, and remove_const traits without <type_traits>.
2. Write a trait is_pointer_to_const detecting if a type is a pointer to a const object.
3. Implement add_pointer and add_reference templates.
4. Write a meta-function is_callable_with<F, Args...> that checks if a callable can be invoked.
5. Implement is_container that detects STL-like containers using SFINAE.
6. Create a meta-function is_specialization_of<T, Template> detecting template instantiations.
7. Implement conditional<B, T, F> yourself.
8. Write a compile-time counter that increments each time a class is instantiated.
9. Implement remove_cvref_t<T> from scratch.
10. Build a trait is_complete<T> to check completeness of a type at compile-time.

⚙️ Category 2 – SFINAE and Detection Idioms

11. Implement has_begin<T> that is true if T has a .begin() member.
12. Write has_value_type<T> using std::void_t.
13. Create a supports_less_than<T> detecting operator <.
14. Write a trait is_streamable<T> to detect if std::cout << T is valid.
15. Implement has_reserve_method<T> using the classic SFINAE pattern (pre–C++17).
16. Write a trait has_iterator_category<T> checking for nested typedef.
17. Build is_invocable_r<R, F, Args...> manually.
18. Implement has_static_member<T, MemberName> detecting static members with name provided as template argument (hard reflection-like).
19. Detect if T is default constructible **without using** <type_traits>.
20. Create a trait that checks if a type T supports operator[].

🧱 Category 3 – Fold Expressions and Parameter Packs

21. Implement a function sum<T...> using fold expressions.
22. Create a compile-time all_true<Booleans...> fold expression.
23. Implement print_all(args...) using pack expansion in a fold expression.
24. Write a fold-based version of logical or_ meta-function.
25. Compute max of a parameter pack of integers at compile time.

26. Write a concat template that joins multiple std::integer_sequences.
27. Implement a compile-time count_types<T, Ts...> returning occurrences of T.
28. Write a fold-based is_any_of<T, Ts...>.
29. Build tuple_cat_type<Tuples...> returning the resulting tuple type.
30. Implement call_all(f, args...) that calls f(args) for each argument using fold expression.

🔧 **Category 4 – Type Transformations and Compile-time Algorithms**

31. Implement unique_types<Ts...> that removes duplicates.
32. Create index_of_type<T, Ts...> returning the index of T in pack.
33. Implement filter_types<Pred, Ts...> returning types satisfying a predicate.
34. Write a meta-function replace_type<From, To, Ts...>.
35. Build zip<Types1..., Types2...> producing pair tuples.
36. Implement a compile-time map<Fn, List> transformation on typelists.
37. Create a meta-function reverse_types<Ts...>.
38. Build a compile-time binary search on a sorted integer_sequence.
39. Implement a meta-function cartesian_product<List1, List2>.
40. Write a merge_sorted_sequences compile-time meta-algorithm.

🧱 **Category 5 – CRTP and Mixin Patterns**

41. Implement Cloneable<T> CRTP that adds a clone() virtual function.
42. Create a CRTP mixin Comparable<T> auto-generating ==, <, > operators.
43. Implement a mixin Printable<T> that provides operator<<.
44. Build a Counter CRTP that tracks instances of a derived type.
45. Implement NamedType<T, Tag> for type-safe strong typedefs.
46. Combine multiple mixins using variadic CRTP inheritance.
47. Write a Serializable<T> CRTP that enforces existence of serialize().
48. Build a CRTP base StaticPolymorphic<T> mimicking runtime polymorphism.
49. Implement Observer<T> and Subject<T> via CRTP.
50. Create a mixin Addable<T> providing operator+ via CRTP.

🧠 **Category 6 – Expression Templates and Lazy Evaluation**

51. Implement a simple expression template for vector arithmetic.
52. Extend it to support chained addition and subtraction.
53. Add scalar multiplication support via expression templates.
54. Build lazy evaluation using templates that defer computation.
55. Implement compile-time expression simplifier (e.g., combine constants).
56. Write a compile-time differentiation engine for arithmetic expressions.
57. Implement an expression tree with eval() resolved at compile-time.
58. Use TMP to optimize redundant expressions in a symbolic algebra system.
59. Build MatrixExpr expression template supporting generic element access.
60. Implement compile-time check to disallow dimension mismatches.

🧱 Category 7 – Compile-time Computation and constexpr TMP

61. Implement compile-time factorial using templates.
62. Implement a compile-time nth_fibonacci.
63. Create a meta-function gcd<A, B> using recursion.
64. Write a compile-time sieve of Eratosthenes generating primes.
65. Implement a constexpr JSON parser using TMP + recursion.
66. Write a compile-time polynomial evaluator.
67. Implement a compile-time string hashing algorithm.
68. Build a compile-time finite state machine.
69. Implement a constexpr parser for arithmetic expressions.
70. Write a compile-time regex matcher using TMP recursion.

⚔️ Category 8 – Concepts and Requires Clauses

71. Define a concept Arithmetic requiring +, -, *, /.
72. Implement a concept Container that requires begin(), end().
73. Create a function template constrained by std::integral.
74. Write your own concept Hashable that enforces std::hash<T>.
75. Implement a concept Comparable requiring == and <.
76. Combine multiple concepts via logical conjunction.
77. Implement a requires clause with nested constraints.

78. Write requires that only enables swap if ADL-resolved swap() exists.
79. Use constrained auto parameters for compile-time function overloading.
80. Implement a mini compile-time dispatcher using constrained templates.

🔁 **Category 9 – Variadic Type Manipulation and Tuple Algorithms**

81. Implement tuple_for_each applying function to each element.
82. Build tuple_transform returning tuple of transformed elements.
83. Implement tuple_zip combining tuples element-wise.
84. Write tuple_filter returning elements satisfying predicate.
85. Build tuple_flatten merging nested tuples.
86. Implement tuple_reverse.
87. Write tuple_apply executing callable with tuple unpacking.
88. Implement tuple_cat_types type-only concatenation.
89. Write a for_each_type<Ts...> compile-time iteration over types.
90. Implement tuple_cartesian_product.

📌 **Category 10 – Building a Mini MPL / Typelist Library**

91. Define a typelist<Ts...> structure.
92. Implement front<List>, back<List>, and pop_front<List>.
93. Build push_front<List, T> and push_back<List, T>.
94. Implement concat<List1, List2>.
95. Create length<List>
96. Build transform<List, MetaFn>.
97. Implement filter<List, Pred>.
98. Write unique<List>.
99. Implement find<List, T>.
100. Build fold<List, Init, Fn> implementing a compile-time reduce.
