Exercise sheet 10 2023-01-19

Due date: 2023-01-26 16:59

The goal of this exercise sheet is to get you used to template creation and compiletime programming.

Exercise 1:

NOTE: To use a struct member at compile time it has to be declared as `constexpr`.

Complete the templated fibonacci struct (`fibonacci.h`) to calculate the fibonacci number for the non-type template parameter **N** that is passed in recursively.

Templates can be called recursively. Use template specializations for the recursive and the base case of the recursion.

Exercise 2:

Complete the compile-time implementation of the **Quadruple** struct (`quadruple.h`) that can store tuples of size 4.

- Add a template specialization for the case where we have four ints. Rather than storing four separate members as in the base template, use a public member `std::array<int, 4>` to store the members.
- Write a partial specialisation for the case where the template arguments refer to the same type. Again, use a public member `std::array` for storing the members.

Exercise 3:

Complete the compile-time implementation of a greatest common divisor computation (`primes.h`) `gcd(...)` which accepts an arbitrary number of arguments (at least two).

- Use variadic templates in order to accept variable numbers of parameters.
- Remember to include a base case to treat an input with two parameters
- Note that the order of the two templates might make a difference - potentially failing to compile even if everything else seems correct.

Exercise 4:

Complete the compile-time implementation of a minimum common multiple computation (`primes.h`) `mcm(...)` which also accepts an arbitrary number of arguments.

It might be useful to use the previous function here.

Exercise 5:

Complete the template metaprogramming implementation of a modular exponentiation computation (`primes.h`) `Power<int base, int exponent, int modulus>::value` (which results in $k \equiv b^e \pmod{m}$)

Include whichever base case(s) you think make sense.