Exercise sheet 10 2023-01-26

Due date: 2023-02-02 16:59

The goal of this exercise sheet is to let you experience the programmer's perspective of creating variadic templates, using template metaprogramming, compile-time programming and type traits. This allows you to have a deeper insight into the process which takes place behind the interface that the user sees, that is, beyond just specifying arguments to the provided functions or templates.

To experience this, you'll implement a *compiletime key-value map*.

Exercise:

Implement a `constexpr` key-value map (similar to `std::map`/`std::unordered_map`), which accepts arbitrary key and value types in `constexprmap.h`.

All key entries of type K must be ensured to be unique.

- Create template `class CexprMap` with parameters `K` and `V`.
- Implement the constructor; check for duplicates (see `verify_no_duplicates` below)
- Implement map construction by a helper function:
```
constexpr auto mymap =
   create_const_map<K, V>(
      std::pair(1, 2),
      std::pair(3, 4));
```
- Add `private` member methods:
  - `find`: returns the iterator matching `key` from the array or `values.end()` if not found
  - `verify_no_duplicates`: raises `std::invalid_argument` exception if duplicate keys are found
- Add `public` member methods:
  - `size`: entry count
  - `contains(const K &)`: Membership test
  - Getter `get(const K &)` and `operator[]`
    throw `std::out_of_range` if key is not found
  - When you throw something at compiletime, your function is no longer constexpr
  - Hence, compilation fails. we can't test that, though :)
- Now allow map construction by constructor! *To make this work, you probably need to implement a template deduction guide!*
```
constexpr CexprMap mymap {
    std::pair(0, 0),
    std::pair(13, 37),
    std::pair(42, 9001)
};
```