Zadanie: Szablonowa klasa Matrix<T, Rows, Cols> - dziś BEZ PLIKU main.cpp(!)

Część I - Zdefiniuj klasę szablonową (0.5 pkt):

```
template <typename T, size_t Rows, size_t Cols>
class Matrix
```

dane są przechowywane w array (`#include <array>; size_t w #include <cstddef>`):

```
std::array<std::array<T, Cols>, Rows>
```
lub
```
std::array<T, Rows * Cols>
```

Część II – Dostęp do elementów (0.5 pkt)

Zaimplementuj metody (funkcje szablonowe!):

- `T& at(size_t r, size_t c);`
- `const T& at(size_t r, size_t c) const;`

Wymagania:

- rozdział na plik .h i .cpp (w pliku „cpp" zakres to "`Matrix<T, Rows, Cols>::`")
- „at" daje dostęp do elementu na pozycji (r, c),
- sprawdzanie poprawności indeksów (użyj assert):
  np. `assert(r < Rows); #include <cassert>`

Część III – Informacje o rozmiarze (0.5 pkt)

Zaimplementuj metody (mogą być w całości w pliku .h):

- `size_t rows() const;`
- `size_t cols() const;`

Zwracają one odpowiednio liczbę wierszy i kolumn macierzy.

Część IV – Operacje wspólne dla wszystkich typów (1 pkt)

Zaimplementuj następujące metody:

- `void Print()` – wypisuje odpowiednio sformatowaną macierz na ekranie
- `void Fill(const T& value);` – ustawia wszystkie elementy macierzy na podaną wartość,
- `size_t Count(const T& value) const;` – zlicza, ile razy dana wartość występuje w macierzy.

Część V – Operacje na macierzy liczb (1.5 pkt)

Dla macierzy, w których T jest typem liczbowym (int, double), zaimplementuj:

- `T Sum() const;` - Sum zwraca sumę wszystkich elementów,
- `double Mean() const;` - zwraca średnią arytmetyczną,

Metody powinny być dostępne **tylko dla typów liczbowych**; przykład deklaracji:

```
template <typename U = T>
std::enable_if_t<std::is_arithmetic<U>::value, U>
Sum() const;
```

Jak to zrobić?

Krok 1: wykrywanie „czy T jest liczbą"

STL daje nam gotowe narzędzie: funkcję w `std::is_arithmetic<T>::value` w bibliotece `#include <type_traits>` , która zwraca true dla: int, double, float, itd.

Krok 2: ograniczenie metod (std::enable_if)

Deklaracja w klasie:

```
template <typename U = T>
std::enable_if_t<std::is_arithmetic<U>::value, typ_zwracany> Sum() const;
```

Dlaczego U = T?

- bez tego enable_if byłby sprawdzany przy definicji klasy,
- a my chcemy, żeby był sprawdzany **dopiero przy wywołaniu metody**.

Krok 3: implementacja Sum() oraz Mean()

W pliku. cpp oprócz standardowego "`template <typename T, size_t Rows, size_t Cols>`" pamiętaj też o: **`template <typename U>`** (tutaj już bez U = T!)
A sama funkcja powinna być implementowana tak:

```
std::enable_if_t<std::is_arithmetic<U>::value, typ_zwracany> Matrix<T, Rows,
Cols>::Function() const
```

Część VI – Operacje na macierzy znaków (tylko dla jednego typu!) – 1 p

Dla macierzy znaków:

- `Matrix<char, Rows, Cols>`

zaimplementuj metodę:

- `void RandomLetters(); - wypełniającą macierz losowymi literami`

Jak upewnić się, że będzie używana tylko dla macierzy znaków? np:

```
static_assert(std::is_same<T, char>::value,  "Available only for Matrix<char>");
```

Część VII – Przykłady użycia

Twoje rozwiązanie **musi poprawnie obsługiwać** poniższe przypadki.

Macierz liczbowa

```
Matrix<int, 3, 3> m;
m.Fill(1);
m.at(1, 1) = 5;
m.Print();
std::cout << m.Count(1) << "\n";
std::cout << m.Sum() << "\n";
std::cout << m.Mean() << "\n";
```

Macierz znaków

```
Matrix<char, 4, 4> board;
board.at(0,0) = 'C';
board.at(0,1) = 'A';
board.at(0,2) = 'T';
board.at(0,3) = 'S';
board.Print();
board.RandomLetters();
board.Print();
```

Czy wiesz że?

- Można napisać **U sum{};** zamiast **U sum = 0;** żeby zainicjalizować wielkość na zero. **To samo z tablicą:** std::array<T, Rows*Cols> tab**{}**;

- Zamiast składni:

```
for (int i= 0 ;i< Rows; i++) {
    for (int j= 0 ;j< Cols; j++) {
        this->at(i,j) = 0;
    }
}
```

Możesz użyć:

```
for (const auto& row : data) {
    for (auto v : row) {
            v = 0;
    }
   } dla  std::array<std::array<T, Cols>, Rows>
```

lub

Albo w przypadku `std::array<T, Rows * Cols>` po prostu

```
for (const auto& v : data) {

   v = 0;
}
```