Lab 6 – CRTP & Variadic templates

Exercise 1. Variadic functions & fold expressions.

Implement variadic functions:

- **average(a1, a2,…, an)** that computes average of any number of numbers a1, …, an (they of numerical type int, double etc)
- **computeSum(f, x1, x2, x3, …)** it should return f(x1)+f(x2)+f(x3)+…
- **insert(container, x1, x2, x3, …)** - it pushes back x1, x2, x3, … into container.

Hint: *You can use fold expressions.*

Exercise 2. Variadic templates and perfect forwarding.

Implement variadic function showTypes that for each argument prints its type and value (you can use typeid or boost::typeindex::type_id).

Implement class template Proxy that has

- constructor that takes a functor f as an argument (its type is the template parameter) and stores it,
- variadic function template operator() that
  - prints information about arguments (using showTypes function)
  - calls functor f forwarding all arguments.

Define function make_proxy that will deduce type of template argument of Proxy and create its object (or you can define deductions rules for Proxy if you use C++17).

Use cases are in file **proxy.cpp**.

Exercise 3. Mixins

Implement class template Mixins that as parameters takes any number of its base classes. Class Mixins should inherit from all of them.
Implement constructor that allows to initialize class with objects of corresponding types.

```
using RedCircle = Mixins<Red, Circle>;
RedCircle x(Red{}, Circle{3});
using BlueRectangleWithNotes = Mixins<Blue, Rectangle, Note>;
BlueRectangleWithNotes y (Blue{}, Rectangle{3,4}, Note{"Hey"});
```

Exercise 4. Mixins

Implement class template **MultiVector** that as parameters takes any number of types.
It should store elements of that types in separate containers e.g. std::vector.
It should be possible to push_back objects into MultiVector and print it to standard output.

MultiVector<int, string, double> m;
m.push_back(5);
m.push_back(string(“text”));
m.push_back(7);

```
m.push_back(1.2);
m.print();   // [ 5 7 ] [ text ] [ 1.2 ]
```