Metaprogramowanie – zadania kwalifikacyjny

Zadania należy wysłać na adres `michal+www13@zielinscy.org.pl`.

C++

Stwórz szablonowy (`template`) typ `MyPair<A, B>`, który reprezentuje parę (analogicznie do `std::pair<A, B>`).

Wymagania:

- (1 pkt) Pola `first` i `second` odpowiednio typów `A` i `B`
- (1 pkt) Konstruktor `MyPair(A a, B b)` odpowiednio ustawiający pola
- (1 pkt) Konstruktor kopiujący `MyPair(MyPair<X, Y>)`, który działa wtw gdy `X` da się skonwertować na `A`, a `Y` na `B` (wskazówka: SFINAE)
- (1 pkt) Funkcja (poza typem) `make_my_pair(A a, B b)`, która zwraca parę `(a, b)`.
- (1 pkt) Dlaczego `make_my_pair` jest przydatna, jeżeli pełni de facto rolę konstruktora?

Przykładowy test:

```
#include "MyPair.h"
#include <cassert>

int main() {
   MyPair<int, float> f (5, 6.5);
   assert (f.first == 5 && f.second == 6.5);

   auto f1 = make_my_pair(5, 6.5);
   assert (f.first == 5 && f.second == 6.5);

   MyPair<float, float> f2 = f;
}
```

Python

Stwórz funkcję `make_mytuple(x1, x2, ..., xn)`, która zwraca wartość `x`, t.ż. `x.a == x1`, `x.b == x2` etc. `n` może być dowolne, z zakresu 0..26. Innymi słowy, musisz stworzyć krotkę indeksowaną literami, a nie liczbami.

Przykładowy test:

```
>>> from mytuple import make_mytuple
>>> t = make_mytuple(1, 2, 3)
>>> assert t.a == 1 and t.b == 2 and t.c == 3
```

```
>>> t.d # ERROR!
>>> from mytuple import make_mytuple
>>> t = make_mytuple()
>>> t.a # ERROR!
>>> t = make_mytuple('foo', 14)
>>> t.a
'foo'
>>> t.b
14
>>>
```

- (2 pkt) Dowolna działająca implementacja
- (1 pkt) Implementacja wykorzystująca `exec`
- (1 pkt) Implementacja wykorzystująca `__getattr__`
- (1 pkt) Pomosłowa implementacja, nie wykorzystująca `exec` ani `__getattr_` (ani podobnych funkcji)

(Aby uzyskać wszystkie punkty, musisz oddać przynajmniej trzy różne implementacje.)