Zadanie L - Prosta para

Do zaimplmentowania jest klasa *DataPair* w pliku *solution.cpp*. Każdy z obiektów tej klasy służy do przechowywania dwóch obiektów klasy *Data*. Niestety definicja klasy *Data* nie jest Ci znana (choć będzie dostępna w czasie testów).

Klasa *DataPair* ma posiadać:

- konstruktory biorące dwa elementy typu *Data* (również przez rvalue reference i w różnych konfiguracjach)
- metody first i second zwracające const'owe referencje do przechowywanych obiektów typu *Data*
- konstruktor kopiujący, move constructor
- operatory przypisania

Klasa przechowuje swoją zawartość w *unique_ptr* tzn. klasa posiada tylko dwa pola zdeklarowane tak:

```
private:
    unique_ptr<Data> first_, second_;
```

Implementacja ma, dodatkowo, spełniać poniższe wymagania:

- wszystkie konieczne kopiowania/przesunięcia są dokonywane jak najmniejszą ilością wywołań konstruktorów, oraz że konstruktory są odpowiednie (np. nie kopiujemy obiektu *Data* jeśli można go przesunąć)
- klasa posiada **strong exception guarantee** tzn. że jeśli operacja się nie powiedzie to stan programu nie ulega zmianie (zakładamy że klasa *Data* daje takie gwarancje)
- operator= jest zaimplementowany używając copy-and-swap
- wszystkie metody nie zmieniające stanu są *const*'owe
- wszystkie metody nie wyrzucające wyjątków są *noexcept* (np. jeśli konstruktor kopiujący dla *Data* jest *noexcept*, to nasz konstruktor kopiujący też ale jeśli nie, to nie)

Dla przykładu kod

```
int main() {
    Data a,b,c,d,e,f;
    cout << "1." << endl;
    DataPair(move(b), move(a));
    cout << "2." << endl;
    DataPair p(c,move(d));
    cout << "3." << endl;
    DataPair q(e,f);
    cout << "4." << endl;
    DataPair r(p);
    cout << "5." << endl;
```

```
    DataPair s(move(q));

    cout << boolalpha << "copy constructor is noexcept? " << noexcept(DataPair(p)) << endl;
    return 0;
}
```

wypisuje, w przypadku kiedy konstruktor kopiujący i przesuwający w klasie *Data* nie rzucają wyjątków, to:

```
constructor()
constructor()
constructor()
constructor()
constructor()
constructor()
1.
constructor(TestingData &&)
constructor(TestingData &&)
2.
constructor(const TestingData &)
constructor(TestingData &&)
3.
constructor(const TestingData &)
constructor(const TestingData &)
4.
constructor(const TestingData &)
constructor(const TestingData &)
5.
copy constructor is noexcept? true
```

a w przypadku kiedy (oba) mogą rzucać to:

```
constructor()
constructor()
constructor()
constructor()
constructor()
constructor()
1.
constructor(const TestingData &)
constructor(const TestingData &)
2.
constructor(const TestingData &)
constructor(TestingData &&)
3.
constructor(const TestingData &)
constructor(const TestingData &)
4.
constructor(const TestingData &)
constructor(const TestingData &)
```

```
5.
copy constructor is noexcept? false
```

W obu przypadkach klasa *Data* wypisuje tylko jakie konstruktory/metody są użyte.
Uwaga :

- wszystkie testy są kompilowane z opcjami *-fno-elide-constructors -fno-inline -Wall -Werror*
- przydatne będzie poczytanie o 'type_traits'.