8 stycznia 2022

Assignment 5

Stwórz wzorzec klasy **Pair**, obiekty której będą reprezentować pary obiektów różnych typów

```
template <typename F, typename S>
class Pair {
    F fst;
    S scn;
public:
    // ...
};
```

F i **S** oznaczają tu typy pierwszego i drugiego elementu pary. Wzorzec powinien definiować:

- konstruktor domyślny;
- konstruktor pobierający referencje do dwóch obiektów dowolnych typów konwertowalnych do typów elementów tworzonego obiektu;
- konstruktor kopiujący, pobierający referencję do innego obiektu-pary, ale z typami elementów niekoniecznie identycznymi z typami elementów tworzonego obiektu — wystarczy, że są do nich konwertowalne;
- operator przypisania — znów typy elementów pary po prawej stronie nie muszą być dokładnie takie same, jak typy elementów pary po lewej stronie przypisania;
- metody **first** i **second** zwracające elementy pary (przez **const** referencję).

Napisz także przeciążony **operator<<** dla obiektów-par.

Stwórz, jako funkcję (szablon) globalną przeciążenie operatora < dla obiektów-par tego samego typu. Pary można porównywać leksykograficznie: najpierw według pierwszego elementu, potem, jeśli to nie daje rozstrzygnięcia, drugiego. Zakładamy, że dla typów **F** i **S** **operator<** działa prawidłowo, ale *nie* zakładamy nic o pozostałych operatorach, jak <=, == itd.

Napisz, jako funkcje globalne, przeciążenia pozostałych operatorów porównujących obiekty-pary: <=, >, >=, == i !=. Wszystkie one powinny być jednolinijkowe i wykorzystywać wyłącznie poprzednio zdefiniowany **operator<**.

Na przykład następujący program

download *Pair.cpp*

```
#include <iostream>

template <typename F, typename S>
class Pair {
    F fst;
    S scn;
```

```
public:
    // ...
};

// ... implementation and other functions

template <typename PAIR>
void check(const PAIR& lhs, const PAIR& rhs) {
    using std::cout; using std::endl;
    auto flags = cout.flags();
    cout << std::boolalpha;
    cout << "\nLHS=" << lhs << "  RHS=" << rhs << endl;
    cout << "lhs <  rhs: " << (lhs <  rhs) << endl;
    cout << "lhs <= rhs: " << (lhs <= rhs) << endl;
    cout << "lhs >  rhs: " << (lhs >  rhs) << endl;
    cout << "lhs >= rhs: " << (lhs >= rhs) << endl;
    cout << "lhs == rhs: " << (lhs == rhs) << endl;
    cout << "lhs != rhs: " << (lhs != rhs) << endl;
    cout.flags(flags);
}

int main () {
    check(Pair<int,int>(3,3), Pair<int,int>(4,2));
    check(Pair<int,int>(3,3), Pair<int,int>(3,2));
    check(Pair<int,int>(4,3), Pair<int,int>(4,3));

    Pair<int,int> pia{3,4};
    Pair<int,int> pib{4,5};
    Pair<double,double> pd(pia);
    std::cout << pia << std::endl;
    std::cout << pd  << std::endl;
    pd = pib;
    std::cout << pd  << std::endl;
}
```

powinien wypisać

```
LHS=[3, 3]  RHS=[4, 2]
lhs <  rhs: true
lhs <= rhs: true
lhs >  rhs: false
lhs >= rhs: false
lhs == rhs: false
lhs != rhs: true

LHS=[3, 3]  RHS=[3, 2]
```

```
lhs <  rhs: false
lhs <= rhs: false
lhs >  rhs: true
lhs >= rhs: true
lhs == rhs: false
lhs != rhs: true

LHS=[4, 3]  RHS=[4, 3]
lhs <  rhs: false
lhs <= rhs: true
lhs >  rhs: false
lhs >= rhs: true
lhs == rhs: true
lhs != rhs: false
[3, 4]
[3, 4]
[4, 5]
```

Nie używaj żadnych dodatkowych dyrektyw **#include**
