#include <iostream>

#include <string>

using namespace std;

class K1 {

string\* p1; // wskaźnik na dwuelementową dynamiczną tablicę stringów

public:

// 1. Konstruktor domyślny

K1() {

p1 = new string[2];

// np. p1[0] = ""; p1[1] = ""; // nie jest to konieczne, ale można

}

// 2. Konstruktor przyjmujący dwie wartości typu string

K1(const string& s1, const string& s2) {

p1 = new string[2];

p1[0] = s1;

p1[1] = s2;

}

// 3. Konstruktor kopiujący

K1(const K1& other) {

p1 = new string[2];

p1[0] = other.p1[0];

p1[1] = other.p1[1];

}

// 4. Destruktor

~K1() {

delete[] p1; // zwalniamy pamięć

}

// 5. Operator przypisania

K1& operator=(const K1& other) {

if (this != &other) {

p1[0] = other.p1[0];

p1[1] = other.p1[1];

}

return \*this;

}

// 6. Operator indeksowania (wersja do zapisu/odczytu)

string& operator[](int i) {

return p1[i];

}

// 7. Operator indeksowania (wersja tylko do odczytu)

const string& operator[](int i) const {

return p1[i];

}

};

class K2 {

K1 dane; // obiekt klasy K1

public:

// 1. Konstruktor domyślny

K2() : dane() {

// nic szczególnego, K1 już ma domyślny konstruktor

}

// 2. Konstruktor przyjmujący dwie wartości typu string

K2(const string& s1, const string& s2) : dane(s1, s2) {

}

// 3. Konstruktor kopiujący

K2(const K2& other) : dane(other.dane) {

}

// 4. Operator przypisania

K2& operator=(const K2& other) {

if (this != &other) {

dane = other.dane;

}

return \*this;

}

// 5. Operator indeksowania -> delegacja do K1

string& operator[](int i) {

return dane[i];

}

const string& operator[](int i) const {

return dane[i];

}

// 6. Operator << (wypisywanie)

friend ostream& operator<<(ostream& os, const K2& obj) {

// Możemy wypisać np. w formacie: "p1[0] p1[1]"

os << obj.dane[0] << " " << obj.dane[1];

return os;

}

};