```cpp
// W_8_0_0.cpp : Defines the entry point for the console application.
//               przeciazenie operatorow +  -  *   /  =

#include "stdafx.h"
#include <iostream>
using namespace std;

class coord
{
    double x;
    double y;
public:
    coord(double xx, double yy) { x = xx; y = yy; }
    coord() { x = 0.0;  y = 0.0; }
    void set(double xx, double yy) { x = xx; y = yy; }
    coord get() { return *this; }
    coord operator + (const coord &praw) const;   //a+b
    coord operator - (const coord &praw) const;   //a-b
    double operator * (const coord &praw) const;  //dot product: dot_prod = a*b
    coord operator * (const double alpa) const;   //a*alpa - mnozenie wektora przez skalar
    coord operator / (const double alpa) const;   //a/alpa - alpa - scalar
    coord &operator = (const coord &praw);  //przypisanie
    void disp(char *tit) { cout << tit << ": " << x << "  " << y << endl; }
};

coord coord::operator + (const coord &praw) const
{
    coord ret;
    ret.x = x+praw.x;
    ret.y = y+praw.y;
    return ret;
}

coord coord::operator - (const coord &praw) const
{
    coord ret;
    ret.x = x-praw.x;
    ret.y = y-praw.y;
    return ret;
}

double coord::operator * (const coord &praw) const
/*=====================================================================
dot_prod
=====================================================================*/
{
    double ret = 0.0;
    ret = x*praw.x+y*praw.y;
    return ret;
}

coord coord::operator * (const double alpa) const
/*=====================================================================
mnozenie wektora przez skalar alpa
=====================================================================*/
{
    coord ret;
    ret.x = x*alpa;
    ret.y = y*alpa;
    return ret;
}

coord coord::operator / (const double alpa) const
/*================================================
dzielenie wektora 2D przez skalar alpa
================================================*/
{
    coord res;
    res.x = x/alpa;
    res.y = y/alpa;
    return res;
}

coord & coord::operator = (const coord &praw)
```

```
/*================================================
Przyciazenie operatora przypisania
zwraca referencje do obiektu
================================================*/
{
    x = praw.x;
    y = praw.y;
    return *this;
}


int _tmain(int argc, _TCHAR* argv[])
{
    coord ob1(2, 3);
    coord ob2(4, 5);
    coord ob3(6, 7);
    coord res;

    ob1.disp("ob1");
    ob2.disp("ob2");
    ob3.disp("ob3");

    //test 1: operator +
    res = ob1+ob2+ob3;
    res.disp("res = ob1+ob2+ob3");

    //test 2: operator -
    res = res-ob2-ob1-ob3;
    res.disp("res = res-ob2-ob1-ob3");

    //test 3: operator * jako iloczyn skalarny
    double dot_prod = ob1*ob2;
    cout << "dot_prod = ob1*ob2 : " << dot_prod << endl;

    //test 4: operator * jako iloczyn skalarny;
    //                  jako mnozenie wektora przez skalar
    res = ob1*(ob2*ob3);
    res.disp("res = ob1*(ob2*ob3)");

    //brak commutativity: pierwszy operand - obiekt klasy coord, a drygi - skalar;
    //                    odwrotnie - nie dziala!
    double scal = 10.0;
    //res = scal*ob1;   //blad kompilacji
    ob1.disp("ob1");
    res = ob1*scal;     //OK
    res.disp("res = scal*ob1");

    //test 5: operator / jako dzielenie przez skalar
    //brak commutativity: pierwszy operand - obiekt klasy coord, a drygi - skalar;
    res = (ob1+ob2)/2.0;
    res.disp("(ob1+ob2)/2");

    res = ob1 = ob2 = ob3;
    res.disp("res: res = ob1 = ob2 = ob3");
    ob1.disp("ob1: res = ob1 = ob2 = ob3");
    ob2.disp("ob2: res = ob1 = ob2 = ob3");
    ob3.disp("ob3: res = ob1 = ob2 = ob3");

    system("pause");
    return 0;
}
```