ZADANIA DO SAMODZIELNEGO ĆWICZENIA

Ćwiczenie 0

Klasa Complex reprezentuje liczbę zespoloną za pomocą dwóch pól prywatnych na część rzeczywistą i urojoną. Napisz dla klasy Complex 4 metody-operatory implementujące operacje dodawania, odejmowania, mnożenia i dzielenia liczb zespolonych. Użyj poniższego szablonu:

```
#include <iostream>

class Complex
{
private:
        double x;
        double y;
public:
        void setX(double a){x=a;}
        void setY(double a){y=a;}
        double getX() const {return x;}
        double getY() const {return y;}
        Complex(double a=0, double b=0) : x(a),y(b){}
        friend std::ostream& operator<<(std::ostream &out, const Complex& c);
        //operatory tutaj
};

std::ostream& operator<<(std::ostream &os, const Complex& c)
{
        os << c.getX() << " +i(" << c.getY() << ")";
        return os;
};



int main()
{
        Complex a(3.0, 6.0);
        Complex b(2.3, 1.0);
        std::cout << (a+b) << " "
                << (a-b) << " "
                << (a*b) << " "
                << (a/b) << std::endl;
        return 0;
}
```

Output: 5.3 +i(7) 0.7 +i(5) 0.9 +i(16.8) 2.05087 +i(1.71701)

Ćwiczenie 1

Zrób to samo co w Ćwiczeniu 0, ale tym razem operatory niech będą zdefiniowane jako funkcje globalne a nie metody.

Ćwiczenie 2

Uzupełnij poniższy program definiując zmienne: signChange, multByKSquared, sumSquared oraz write, będące funkcjami lambda. Funkcja signChange powinna zmienić znak elementom wektora v. Funkcja multByKSquared powinna przyjąć jeden argument typu int, liczbę k, i przemnożyć wszystkie elementy wektora v przez k^2. Funkcja sumSquared zwraca sumę kwadratów elementów wektora v. Ta funkcja nie powinna mieć możliwości zmiany wektora v, czyli nie należy używać referencji tylko przekazać v przez kopię. Ostatnia funkcja, write, wypisuje wektor na standardowe wyjście.

```
#include <vector>
#include <iostream>

using namespace std;

int main()
{
        vector<int> v= {1,-4, 2, 1, 0};

        auto signChange = ...
        auto multByKSquared = ..
        auto sumSquared  =  ...
        auto write =...


        write();
        cout << sumSquared() << endl;
        multByKSquared(2);
        signChange();
        write();
        return 0;
}
```

Output:
1 -4 2 1 0
22
-4 16 -8 -4 0