

# OO Programozás

Operátor felültöltés, másolás és átalakítás

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

[https://github.com/wajzy/GKxB\\_INTM085](https://github.com/wajzy/GKxB_INTM085)

2023. október 17.

Jelenlegi legjobb tudásunk szerint készítsünk osztályt egy **komplex szám** tárolására és műveletek elvégzésére, azaz:

- tároljuk a szám valós és képzetes részét,
- készítsünk konstruktorokat,
- getter/setter tagfüggvényeket,
- és olyan függvényeket, melyekkel komplex számok összeadhatók és szorozhatók!

## Complex1.h

```
6  class Complex {
7      double re, im;
8  public:
9      Complex() {
10         re = im = 0.;
11     }
12     Complex(double re, double im) {
13         this->re = re;
14         this->im = im;
15     }
16     double getRe() {
17         return re;
18     }
19     void setRe(double re) {
20         this->re = re;
21     }
```

## Complex1.h

```
22     double getIm() {
23         return im;
24     }
25     void setIm(double im) {
26         this->im = im;
27     }
28     void print() {
29         std::cout << re << '+' << im << 'i' << std::endl;
30     }
31     Complex add(Complex right) {
32         return Complex(re + right.re, im + right.im);
33     }
34     Complex mult(Complex right) {
35         return Complex(re*right.re - im*right.im, im*right.re + re*right.im);
36     }
37 };
```

## main1.cpp

```
1 #include <iostream>
2 #include "Complex1.h"
3
4 int main() {
5     Complex c(1., 1.);
6     Complex sum = c.add(Complex(2., 2.));
7     sum.print(); // 3+3i
8     Complex total = c.mult(Complex(3., 4.));
9     total.print(); // -1+7i
10 }
```

## Operátor felültöltés

- A C++ megengedi, hogy az operátorok jelentését kiterjesszük a saját típusainkra (osztályainkra)
- Pl. ha értelmezhető az összeadás két int vagy float között, akkor két Complex objektum miért ne lehetne összeadható?
- Az operátorok működését (praktikusan nyilvános tag)függvények adják meg → `operatorX`, ahol X pl. `+`, `*`.

## Complex2.h

```
31 // Use reference to avoid copy of 'right'
32 Complex operator+(const Complex& right) {
33     return Complex(re + right.re, im + right.im);
34 }
35 Complex operator*(const Complex& right) {
36     return Complex(re*right.re - im*right.im, im*right.re + re*right.im);
37 }
```

- + művelet bal operandusa: aktuális objektum, a jobb oldalt paraméterként kapja.
- Utóbbi felesleges (tagonkénti) másolásának elkerülésére referenciát használunk.
- Új, ideiglenes (eredmény) objektum jön létre e kettő alapján, ezt adja vissza.

## main2.cpp

```
4  int main() {
5      Complex c(1., 1.);
6      // Complex sum = c.operator+(Complex(2., 2.));
7      Complex sum = c + Complex(2., 2.);
8      sum.print();
9      // Complex sum2 = c + 100.;
10     // error: no match for 'operator+'
11     // (operand types are 'Complex' and 'double')
12     // note: no known conversion for argument 1
13     // from 'double' to 'const Complex&'
14     Complex total = c * Complex(3., 4.);
15     total.print();
16 }
```



### Probléma:

Összeadásnál a jobb oldali operandusnak Complex-nek kell lennie.

### Megoldás:

További felültöltött operátor függvények hozzáadása, pl. double-t hozzáadhatunk a valós részhez.

#### Complex3.h

```
32     Complex operator+(double re) {  
33         return Complex(this->re + re, im);  
34     }
```

## main3.cpp

```
4  int main() {  
5      Complex c(1., 1.);  
6      // Complex sum = c.operator+(Complex(2., 2.));  
7      Complex sum = c + Complex(2., 2.);  
8      sum.print();  
9      // Complex sum2 = c.operator+(100.);  
10     Complex sum2 = c + 100.;  
11     sum2.print();  
12     // Complex sum3 = 100. + c;  
13     // error: no match for 'operator+'  
14     // (operand types are 'double' and 'Complex')
```

- Ez sem segít, ha a *bal* operandus double típusú → *nem tag*, két paraméteres operator függvény.
- Nem éri el a privát/védett tagokat → barát (friend) függvény: mindenhez hozzáfér az osztályon belül.
- Tagobjektum is lehet az osztályunk barátja, pl.: `friend class FriendOfComplex;`

## Complex4.h

```
6  class Complex {
7      double re, im;
8      public:

36      friend std::ostream& operator<<(std::ostream& os, const Complex& cplx);
37
38      Complex operator+(const Complex& right) {
39          return Complex(re + right.re, im + right.im);
40      }
41      Complex operator+(double re) {
42          return Complex(this->re + re, im);
43      }
44      friend Complex operator+(double re, const Complex& right);
```

## Complex4.cpp

```
3 std::ostream& operator<<(std::ostream& os, const Complex& c) {  
4     os << c.re << '+' << c.im << 'i';  
5     return os;  
6 }  
7  
8 Complex operator+(double re, const Complex& right) {  
9     return Complex(re + right.re, right.im);  
10 }
```