Napredno programiranje i programski jezici

03 C++ (preklapanje operatora)

Fakultet tehničkih nauka, Novi Sad
23-24/Z
Dunja Vrbaški

Prethodno predavanje
konstruktori, destruktori, problemi

U nastavku:
preklapanje operatora

```
// double: re i im

class Complex {

private:
    ...

public:

    ...
};
```

```
// double: re i im

class Complex {

private:
    ...

public:


    ...
};
```

```
int x, y, z;
z = x + y;

double a, b, c;
c = a * b;

Complex z1, z2, z3;
z3 = z1 + z2;
```

Kako da obezbedimo da aritmetički operatori mogu da se primene i na objekte naše klase?

```
class Complex {

private:
   double real;
   double imag;
public:
   Complex();
   Complex(double, double);
   Complex(const Complex&);

   double getReal() const;
   double getImag() const;
   void setReal(double);
   void setImag(double);

   // operator +...
   // operator *...
   // operator =...
   ...
};
```

```
int x, y, z;
z = x + y;

double a, b, c;
c = a * b;

Complex z1, z2, z3;
z3 = z1 + z2;
```

```
class Complex {

private:
   double real;
   double imag;
public:
   Complex();
   Complex(double, double);
   Complex(const Complex&);

   double getReal() const;
   double getImag() const;
   void setReal(double);
   void setImag(double);

   // operator +...
   // operator *...
   // operator =...
   ...
};
```

Ne mogu se redefinisati za std tipove (int…)
Ne mogu se uvoditi novi
Ne mogu se definisati operatori:

- za pristup članu klase .
- za razrešenje dosega ::
- za uslovni izraz ?:
- za veličinu objekta sizeof

```
class Complex {

private:
   double real;
   double imag;
public:
   Complex();
   Complex(double, double);
   Complex(const Complex&);

   double getReal() const;
   double getImag() const;
   void setReal(double);
   void setImag(double);

   // operator +...
   // operator *...
   // operator =...
   ...
};
```

metod

ILI

slobodna funkcija

PRIJATELJSKE FUNKCIJE

```
class MyClass {

private:
   int x;
public:
    ...
};
```

```
void foo(const MyClass &mc) {
   ~~int a = mc.x;~~
}

int main()
{
   MyClass mc;
   ~~int a = mc.x;~~
}
```

```
class MyClass {

private:
   int x;
public:
    ...

    friend foo(const MyClass&);
};
```

```
void foo(const MyClass &mc) {
   int a = mc.x;
}

int main()
{
   MyClass mc;
   ~~int a = mc.x;~~
}
```

```
class Complex {

private:
   double real;
   double imag;
public:
   Complex();
   Complex(double, double);
   Complex(const Complex&);

   double getReal() const;
   double getImag() const;
   void setReal(double);
   void setImag(double);

   // operator +...
   // operator *...
   // operator =...
   ...
};
```

metod ILI slobodna funkcija

Neki se moraju definisati kao metode
Neki mogu kao metod ili kao funkcija

```
class Complex {

private:
   double real;
   double imag;
public:

   Complex& operator=(const Complex&);

   friend Complex operator+(const Complex&, const Complex&);

   ...
};
```

complex.hpp

Neki se moraju definisati kao metode
Neki mogu kao metod ili kao funkcija (slobodna ili ne)

```
class Complex {

private:
   double real;
   double imag;
public:

   Complex& operator=(const Complex&);

   friend Complex operator+(const Complex&, const Complex&);

   ...
};
```

```
z1 = z2                 z1.operator=(z2)
z1 + z2                 operator+(z1, z2)
```

```
Complex& operator=(const Complex&);
Complex& operator+=(const Complex&);
Complex& operator-=(const Complex&);
Complex& operator*=(const Complex&);
Complex& operator/=(const Complex&);
const Complex& operator++();
const Complex operator++(int);
```

```
friend Complex operator+(const Complex&, const Complex&);
friend Complex operator-(const Complex&, const Complex&);
friend Complex operator*(const Complex&, const Complex&);
friend Complex operator/(const Complex&, const Complex&);
friend bool operator==(const Complex&, const Complex&);
friend bool operator!=(const Complex&, const Complex&);
friend ostream& operator<<(ostream&, const Complex&);
friend istream& operator>>(istream&, Complex&);
```

```
Complex& Complex::operator=(const Complex &z){ ... }

...

Complex operator+(const Complex &z1, const Complex &z2) { … }
```

complex.cpp

```
Complex& Complex::operator=(const Complex &z) {
   real = z.real; imag = z.imag;
   return *this;
}
```

- u ovom primeru, kao implicitni operator dodele
- this - pokazivač na sam objekat
- dereferencira se i vraća se referenca na objekat
  `(z1 = z2 = z3)`

```
Complex& Complex::operator+=(const Complex &z) {
   real += z.real; imag += z.imag;
   return *this;
}
```

```
Complex& Complex::operator-=(const Complex &z) {
   real -= z.real; imag -= z.imag;
   return *this;
}
```

```
Complex& Complex::operator*=(const Complex &z) {
   double r = real * z.real - imag * z.imag;
   double i = real * z.imag + imag * z.real;
   real = r; imag = i;
   return *this;
}
```

```
Complex& Complex::operator/=(const Complex &z) {
   double r = real;
   double i = imag;
   double d = z.real * z.real + z.imag * z.imag;
   real = (r * z.real + i * z.imag) / d;
   imag = (i * z.real - r * z.imag) / d;
   return *this;
}
```

```
const Complex& Complex::operator++() {
   real++; imag++;
   return *this;
}
```

```
const Complex Complex::operator++(int i) {
   Complex w(real, imag);
   real++; imag++;
   return w;
}
```

Postfiksni (z++) i prefiksni (++z). Koji je koji?

```
Complex operator+(const Complex &z1, const Complex &z2) {
   Complex w(z1.real + z2.real, z1.imag + z2.imag);
   return w;
}
```

```
Complex& Complex::operator+=(const Complex &z) {
   real += z.real; imag += z.imag;
   return *this;
}
```

```
friend Complex operator+(const Complex&, const Complex&);
```

```
Complex operator+(const Complex &z1, const Complex &z2) {
   Complex w(z1.real + z2.real, z1.imag + z2.imag);
   return w;
}
```

Da funkcija nije bila deklarisana kao friend u klasi morali bismo da koristimo get/set.

~~`friend Complex operator+(const Complex&, const Complex&);`~~

```
Complex operator+(const Complex &z1, const Complex &z2) {
   Complex w(z1.getReal() + z2.getReal(), z1.getImag() + z2.getImag());
   return w;
}
```

```
Complex operator-(const Complex &z1, const Complex &z2){
   Complex w(z1.real - z2.real, z1.imag - z2.imag);
   return w;
}
```

```
Complex operator*(const Complex &z1, const Complex &z2){
   Complex w(z1.real * z2.real - z1.imag * z2.imag,
   z1.imag * z2.real + z1.real * z2.imag);
   return w;
}
```

```
Complex operator/(const Complex &z1, const Complex &z2){
   double d = z2.real * z2.real + z2.imag * z2.imag;
   Complex w((z1.real * z2.real + z1.imag * z2.imag) / d,
             (z1.imag * z2.real - z1.real * z2.imag) / d);
   return w;
}
```

```
bool operator==(const Complex &z1, const Complex &z2){
    return (z1.real == z2.real) && (z1.imag == z2.imag);
}
```

```
bool operator!=(const Complex &z1, const Complex &z2){
    return (z1.real != z2.real) || (z1.imag != z2.imag);
}
```

```
ostream& operator<<(ostream &out, const Complex &z){
   if( z.imag == 0)
      out << z.real;
   if( z.real == 0 && z.imag != 0)
      out << z.imag << "i";
   if( z.real != 0 && z.imag > 0)
      out << z.real << "+" << z.imag << "i";
   if( z.real != 0 && z.imag < 0)
     out << z.real << z.imag << "i";
   return out;
}
```

```
istream& operator>>(istream &in, Complex &z){
   in >> z.real >> z.imag;
   return in;
}
```

```
int main() {
   Complex z1, z2(1,1), z3(2,3);

   z1 = z2 + z3;
   cout <<"z1 = " << z1 << endl;
   cin >> z3;
   if (z1 == z2) { … }
   …
}
```

ZADATAK
(neobavezno)

Isprobati sve iz prezentacije.

Implementirati samostalno klasu Complex.
Testirati sve operatore.