Sobrecarga de Operadores

- Assim como é possível sobrecarregar métodos e funções, também é possível sobrecarregar os operadores de C++
- Em C, a sobrecarga de operadores era implícita
- A aplicação de um operador é equivalente à chamada de uma função
- Em C++ é permitido sobrecarregar um operador, com o objetivo de simplificar a notação e uniformizar a expressão.
- Existem duas maneiras de implementar operadores para classes de objetos:
  - como funções membro
  - como funções globais

Sobrecarga de Operadores: Exemplo

- Para se sobrecarregar operadores é necessário compreender um pouco mais a fundo o comportamento desses operadores sobre os tipos básicos da linguagem.
- Considere o operador unário `!`(NOT) associado a um objeto *w* de uma classe qualquer *W*.
- A expressão !w é equivalente a:
  - w.operator!(); // usando uma função membro
  - operator!(w); // usando uma função global
- Para um operador binário, de soma por exemplo, a expressão *w1 + w2* é equivalente as seguintes chamadas:
  - x.operator+(y); // usando uma função membro
  - Operator+(x,y); // usando uma função global

Exemplo - Classe complex

```
class Complex {
   double re, im;
 public:
   Complex operator+ (const Complex&);
   Complex operator- (const Complex&);
};

// Com esses operadores, é possível fazer:

void main()
{
   Complex c1, c2, c3;
   // ...
   c1 = c2 + c3;
}
```

- Mas como implementar esses operadores?

Classe complex (funções membro)

```
Complex Complex::operator+(const Complex& c)
{
  Complex s;
  s.re = this.re + c.re; s.im = this.im + c.im;
  return s;
}
Complex Complex::operator-(const Complex& c)
{
  Complex s;
  s.re = this.re - c.re; s.im = this.im - c.im;
  return s;
}
main()
{
  Complex r, s, t, u, v;
  // ...
  r = s + t;  // r = s.operator+(t);
  u = r - v;  // u = s.operator-(v);
}
```

Classe complex (funções globais)

- Se o operador é binário e o método que faz a sua sobrecarga recebe como parâmetros dois objetos, então o método não pode ser chamado partindo-se de um objeto da classe, ou seja, a função é global:

```
Complex Complex::operator-(const Complex& c1, const Complex& c2)
{
  Complex s;
  s.re = c1.re + c2.re; // ERROR: re e im são private
  s.im = c1.im + c2.im;
  return s;
}
```

- Nesse caso o mais usual é declarar a sobrecarga como *friend* da classe.

Classe complex (funções globais)

```
class Complex {
   // ...
   friend Complex operator+ (const Complex&);
   friend Complex operator- (const Complex&);
};
```

- Com isso é possível acessar os campos privados da classe Complex.
- Com as modificações, o uso dos operadores passa a ser equivalente as seguintes chamadas de métodos:

```
main()
{
  Complex r, s, t, u, v;
  // ...
  r = s + t;  // r = operator+(s,t);
  u = r - v;  // u = operator-(s,v);
}
```