

C++

Básico ao Avançado

Sim, você já usou

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

1 Onde usamos

2 Sobrecarregando

3 Metacarregando

4 Este ponteiro

5 Hora de brincar

Usando aqui

- Você se lembra dos irmãos *console*? `cin` e `cout`
- Já se perguntou como os operadores de deslocamento fazem isso? `>>`, `<<`
- Lembra do desafio de somar ∇ ?
- Imagina poder fazer algo como `v1 + v2` com uma `struct`
- Tudo isso é possível com sobrecarregamento de operadores

```
#include <iostream>
int main()
{
    int A;
    std::cin >> A;
    return 0;
}
```

```
#include <iostream>
int main()
{
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

E ainda...

```
#include <iostream>
int main()
{
    bool A;
    char B;
    int C;
    double D;
    std::cin >> A >> B >> C >> D;
    std::cout << A << B << C << D << std::endl;
    return 0;
}
```

- Consiste em alterar a operação do operador
- A sintaxe de sobrecarga é padrão

```
//...  
<tipo0> operator <operador>(<tipo1> <arg1>, ...)  
{  
    //...  
}  
//...
```

- Chatices:
 - Nem todos os operadores são sobrecarregáveis
 - Não pode ser apenas em tipos primitivos
 - Ordem de precedência não pode ser alterada
 - Ordem de agrupamento não pode ser alterada *É o que?*
 - Novos operadores não podem ser criados

Inteiro

```
1  class inteiro
2  {
3  private:
4      int valor;
5  public:
6      inteiro(int n)
7      {
8          valor = n;
9      }
10     int getValor()
11     {
12         return valor;
13     }
14     int operator+(inteiro I)
15     {
16         return getValor()+I.
17             getValor();
18     }
19 };
```

```
19  int main()
20  {
21      inteiro A(5);
22      inteiro B(10);
23      inteiro C(A + B);
24      int D;
25      D = A.operator+(C);
26      return 0;
27  }
```

Retornos estrambólicos

- *Ah, mas eu lembro que os **consoles** aceitavam mais de um...*
- *E eu testei aqui e deu errado...*
- *Tem uma forma de fazer isso?*
- Sim!
- Você pode retornar o próprio objeto!
- *É o que?*

Mind blow

- Podemos retornar variáveis de funções
- Também podemos retornar tipos abstratos com funções
- E podemos retornar objetos com funções do tipo de sua classe
- O código anterior pode ficar mais interessante. . .

Interessantemente

```
1  class inteiro
2  {
3  private:
4      int valor;
5  public:
6      inteiro(int n)
7      {
8          valor = n;
9      }
10     int getValor()
11     {
12         return valor;
13     }
14     inteiro operator+(inteiro I)
15     {
16         return inteiro(getValor()+
17                         I.getValor());
18     };
```

```
19  int main()
20  {
21      inteiro A(5);
22      inteiro B(10);
23      inteiro C(A + B);
24      inteiro D = B + C;
25      B = A + C + D;
26      return 0;
27  }
```

Mas...

- Ficou meio ambíguo...

```
int operator+(inteiro I)
{
    return getValor()+I.getValor();
}
```

```
inteiro operator+(inteiro I)
{
    return inteiro(getValor()+I.getValor());
}
```

- Qual getValue() é de quem?

Este

- Para evitar isso existe um ponteiro especial
- Além de evitar isso, pode fazer mais coisas
- Este ponteiro aponta para o objeto em questão *Em questão?*
- É denotado pela palavra-chave `this`

```
this;  
//...  
this-><membro>;  
this-><membro>();  
//...  
return *this;
```

Novamente...

```
1  class inteiro
2  {
3  private:
4      int valor;
5  public:
6      inteiro(int n)
7      {
8          valor = n;
9      }
10     int getValor()
11     {
12         return valor;
13     }
14     inteiro operator+(inteiro I)
15     {
16         return inteiro(this->getValor()+I.getValor());
17     }
18 };
```

Vamos testar!