C++
Básico ao Avançado
Sim, você já usou

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

Usando aqui

Usando aqui

- Você se lembra dos irmãos *console*?

Usando aqui

- Você se lembra dos irmãos *console*? `cin` e `cout`

Usando aqui

- Você se lembra dos irmãos *console*? `cin` e `cout`
- Já se perguntou como os operadores de deslocamento fazem isso?

Usando aqui

- Você se lembra dos irmãos *console*? `cin` e `cout`
- Já se perguntou como os operadores de deslocamento fazem isso? `>>`,`<<`

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 $\overrightarrow{v}$?

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 $\overrightarrow{v}$?
- Imagina poder fazer algo como `v1 + v2` com uma `struct`

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 $\overrightarrow{v}$?
- Imagina poder fazer algo como `v1 + v2` com uma `struct`
- Tudo isso é possíve com sobrecarregarga de operadores

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 $\overrightarrow{v}$?
- Imagina poder fazer algo como `v1 + v2` com uma `struct`
- Tudo isso é possíve com sobrecarregarga de operadores

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

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 $\overrightarrow{v}$?
- Imagina poder fazer algo como `v1 + v2` com uma `struct`
- Tudo isso é possíve com sobrecarregarga 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...

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

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

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

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

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

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

- Chatices:

- 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

- 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

- 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

- 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

- 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?*

- 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

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.
           getValor();
17   }
18 };
```

```
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

Retornos estrambólicos

- *Ah, mas eu lembro que os* **consoles** *aceitavam mais de um...*

Retornos estrambólicos

- *Ah, mas eu lembro que os* **consoles** *aceitavam mais de um...*
- *E eu testei aqui e deu errado...*

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?*

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!

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!

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

Mind blow

- Podemos retornar variáveis de funções

Mind blow

- Podemos retornar variáveis de funções
- Também podemos retornar tipos abstratos com funções

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

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

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()+
           I.getValor());
17   }
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. . .

Mas...

- Ficou meio ambíguo...

Mas...

- Ficou meio ambíguo...

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

Mas...

- Ficou meio ambíguo...

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

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

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

Este

- Para evitar isso existe um ponteiro especial

Este

- Para evitar isso existe um ponteiro especial
- Além de evitar isso, pode fazer mais coisas

Este

- Para evitar isso existe um ponteiro especial
- Além de evitar isso, pode fazer mais coisas
- Este ponteiro aponta para o objeto em questão

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?*

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 paravra-chave `this`

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 paravra-chave `this`

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

Novamente. . .

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!