C++
Básico ao Avançado
Nasce e morre

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

Inicializando

Inicializando

- Precisamos inicializar atributos

Inicializando

- Precisamos inicializar atributos
- Precisamos alocar memória

Inicializando

- Precisamos inicializar atributos
- Precisamos alocar memória
- Exibir informação sobre estado de inicialização

Inicializando

- Precisamos inicializar atributos
- Precisamos alocar memória
- Exibir informação sobre estado de inicialização
- Fazer cálculos préviamente de valor fixos

Inicializando

- Precisamos inicializar atributos
- Precisamos alocar memória
- Exibir informação sobre estado de inicialização
- Fazer cálculos préviamente de valor fixos
- Em alguns casos, *casting*

Inicializando

- Precisamos inicializar atributos
- Precisamos alocar memória
- Exibir informação sobre estado de inicialização
- Fazer cálculos préviamente de valor fixos
- Em alguns casos, *casting* (outra aula)

Construtor

Construtor

- Recebe o nome da classe

Construtor

- Recebe o nome da classe
- Não tem tipo

Construtor

- Recebe o nome da classe
- Não tem tipo, nem mesmo `void`

Construtor

- Recebe o nome da classe
- Não tem tipo, nem mesmo `void`
- Pode receber argumentos

Construtor

- Recebe o nome da classe
- Não tem tipo, nem mesmo `void`
- Pode receber argumentos
- É invocado automaticamente

Construtor

- Recebe o nome da classe
- Não tem tipo, nem mesmo `void`
- Pode receber argumentos
- É invocado automaticamente
- Deve ser `public`

Construtor

- Recebe o nome da classe
- Não tem tipo, nem mesmo `void`
- Pode receber argumentos
- É invocado automaticamente
- Deve ser `public`

```
class <nome>
{
  //...
public:
  <nome>(<tipo1> <arg1>, ... , <tipoN> <argN>)
  {
    //...
  }
  //...
};
```

Encerrando

Encerrando

- *Encerrar atributos*

Encerrando

- *Encerrar atributos???*

Encerrando

- *Encerrar atributos???*
- E se os atributo forem objetos?

Encerrando

- *Encerrar atributos???*
- E se os atributo forem objetos?
- Desalocar memória

Encerrando

- *Encerrar atributos???*
- E se os atributo forem objetos?
- Desalocar memória
- Exibir informações sobre estado de encerramento

Encerrando

- *Encerrar atributos???*
- E se os atributo forem objetos?
- Desalocar memória
- Exibir informações sobre estado de encerramento
- *Desfazer cálculos*

Encerrando

- *Encerrar atributos???*
- E se os atributo forem objetos?
- Desalocar memória
- Exibir informações sobre estado de encerramento
- *Desfazer cálculos???*

Encerrando

- *Encerrar atributos???*
- E se os atributo forem objetos?
- Desalocar memória
- Exibir informações sobre estado de encerramento
- *Desfazer cálculos???*
- A ideia de *desfazer cálculos* é semelhante à ideia de limpar memória para desalocar

Encerrando

- *Encerrar atributos???*
- E se os atributo forem objetos?
- Desalocar memória
- Exibir informações sobre estado de encerramento
- *Desfazer cálculos???*
- A ideia de *desfazer cálculos* é semelhante à ideia de limpar memória para desalocar
- É padrão tratar a memória quando se aloca, por esperar ela suja

Encerrando

- *Encerrar atributos???*
- E se os atributo forem objetos?
- Desalocar memória
- Exibir informações sobre estado de encerramento
- *Desfazer cálculos???*
- A ideia de *desfazer cálculos* é semelhante à ideia de limpar memória para desalocar
- É padrão tratar a memória quando se aloca, por esperar ela suja
- Pra que limpar...

Destrutor

Destrutor

- Recebe o nome da classe

Destrutor

- Recebe o nome da classe com um til (~)

Destrutor

- Recebe o nome da classe com um til (~)
- Não tem tipo

Destrutor

- Recebe o nome da classe com um til (~)
- Não tem tipo, nem mesmo `void`

Destrutor

- Recebe o nome da classe com um til (~)
- Não tem tipo, nem mesmo `void`
- Não pode receber argumentos

Destrutor

- Recebe o nome da classe com um til (~)
- Não tem tipo, nem mesmo `void`
- Não pode receber argumentos
- É invocado automaticamente

Destrutor

- Recebe o nome da classe com um til (~)
- Não tem tipo, nem mesmo `void`
- Não pode receber argumentos
- É invocado automaticamente
- Deve ser `public`

Destrutor

- Recebe o nome da classe com um til (~)
- Não tem tipo, nem mesmo `void`
- Não pode receber argumentos
- É invocado automaticamente
- Deve ser `public`

```
class <nome>
{
  //...
public:
  ~<nome>()
  {
    //...
  }
  //...
};
```

Compara

```
class <nome>
{
  //...
public:
  <nome>(<tipo1> <arg1>, ... , <tipoN> <argN>)
  {
    //...
  }
  //...
};
```

Compara

```
class <nome>
{
  //...
public:
  ~<nome>()
  {
    //...
  }
  //...
};
```

Nasce e morre

Nasce e morre

```
1  class alpha
2  {
3  private:
4    int N, *P;
5  public:
6    alpha(int n)
7    {
8      N = n;
9      P = new int[N];
10   }
11   ~alpha()
12   {
13     delete[] P;
14   }
15 };
16
17 int main()
18 {
19   alpha A(10);
20 }
```

Vamos testar!