Programação Orientada a Objetos

PONTEIRO «this»

em C++

Introdução

- **Construtores** permitem a inicialização de objetos
  - São chamados na criação do objeto
    - É recomendado fornecer pelo menos um
    - Pode existir mais de um

```
Jogo::Jogo()
{
    nome  = "vazio";
    preco = 0;
    horas = 0;
    custo = 0;
}
```

```
Jogo::Jogo(const string & titulo, float valor)
{
    nome  = titulo;
    preco = valor;
    horas = 0;
    custo = valor;
}
```

Introdução

- **Destrutores** permitem a liberação de recursos
  - São chamados no fim da vida do objeto
    - Devem ser fornecidos para:
      - Liberar memória
      - Encerrar conexões
      - Fechar arquivos

Essenciais para o gerenciamento de memória

```
Conjunto::Conjunto(int n)
{
    // aloca memória
    vet = new int[n];
}
```

```
Conjunto::~Conjunto()
{
    // libera memória
    delete [] vet;
}
```

Ponteiro «this»

- Como os métodos acessam atributos?

```
void Jogo::exibir()
{
    cout << nome   << " R$"
         << preco  << " "
         << horas  << "h = R$"
         << custo  << "/h\n";
}
```

Métodos possuem acesso implícito aos atributos

O compilador cuida disso: eles estão no escopo da classe, certo?

Sim, mas ...

Ponteiro «this»

- A primeira implementação de C++ usava C com classes
  - Ela convertia métodos em funções

```
void Jogo ::exibir()
{
    cout <<  nome   << " R$"
         <<  preco  << " "
         <<  horas  << "h = R$"
         <<  custo  << "/h\n";
}
```

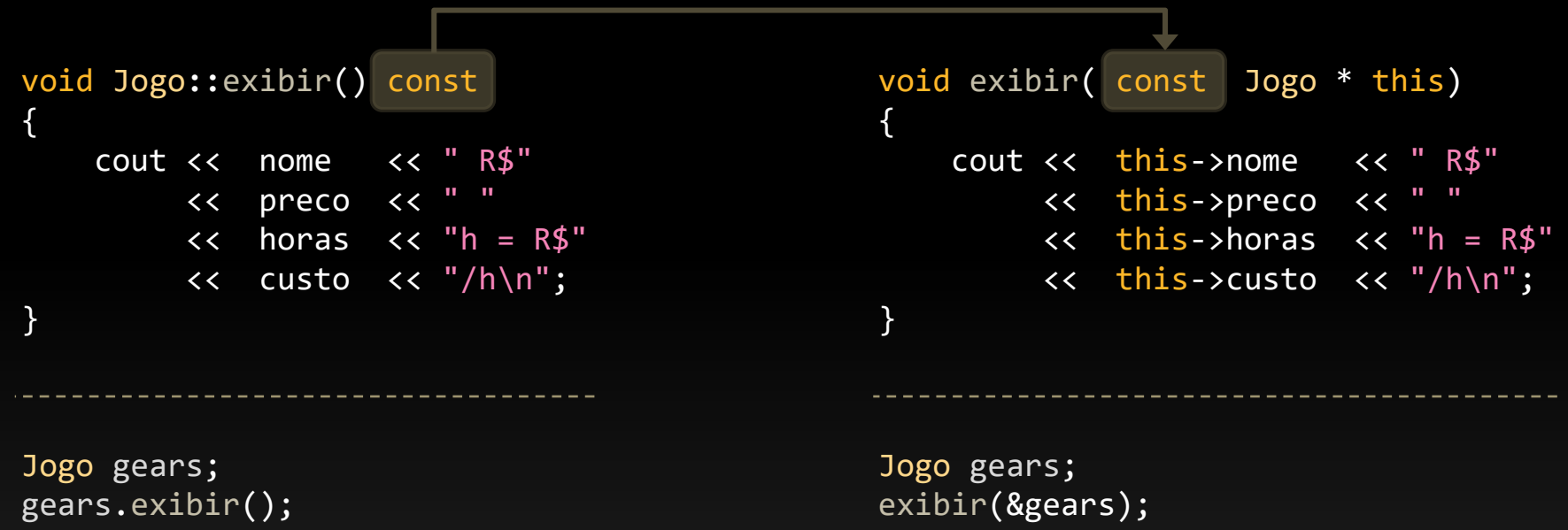
```
Jogo gears;
gears.exibir();
```

```
void exibir( Jogo * this)
{
    cout <<  this->nome   << " R$"
         <<  this->preco  << " "
         <<  this->horas  << "h = R$"
         <<  this->custo  << "/h\n";
}
```

```
Jogo gears;
exibir(&gears);
```

Ponteiro «this»

- Um método const não pode modificar os atributos
  - Porque ele é traduzido em um ponteiro constante



```
void Jogo::exibir() const
{
    cout << nome   << " R$"
         << preco  << " "
         << horas  << "h = R$"
         << custo  << "/h\n";
}

---

Jogo gears;
gears.exibir();
```

```
void exibir( const Jogo * this)
{
    cout << this->nome   << " R$"
         << this->preco  << " "
         << this->horas  << "h = R$"
         << this->custo  << "/h\n";
}

---

Jogo gears;
exibir(&gears);
```

Ponteiro «this»

- Os compiladores atuais não traduzem C++ em C
  - Mas o ponteiro this continua existindo
    - Ele é passado implicitamente aos métodos
    - Aponta para o objeto usado na chamada

```
void Jogo::exibir() const
{
    cout <<  this->nome   << " R$"
         <<  this->preco  << " "
         <<  this->horas  << "h = R$"
         <<  this->custo  << "/h\n";
}
```

```
Jogo gears { "Gears", 90.0f };

gears.atualizar(100.0f);
gears.jogar(2);
gears.exibir();
```

Usos do Ponteiro «this»

- Fazer referência aos atributos do objeto
  - Diferenciar nomes de parâmetros e atributos

```
class Jogo
{
private:
   string nome;
   float preco;
   int horas;
   float custo;

   ...
};
```

```
Jogo::Jogo(const string & nome, float preco)
{
    this->nome  = nome;   ✓
    this->preco = preco;  ✓
    horas = 0;
    custo = preco;
}
```

Usos do Ponteiro «this»

- Fazer referência ao próprio objeto
  - Imagine ter métodos para comparar objetos
    - Obter o jogo com mais horas jogadas
    - Obter o jogo com melhor custo

```
const Jogo & Jogo::comparar(const Jogo & jogo) const;
-------------------------------------------------------------

Jogo gears { "Gears", 90.0f };
Jogo doom { "Doom", 60.0f };

-------------------------------------------------------------

const Jogo & top = gears.comparar(doom);
```

Usos do Ponteiro «this»

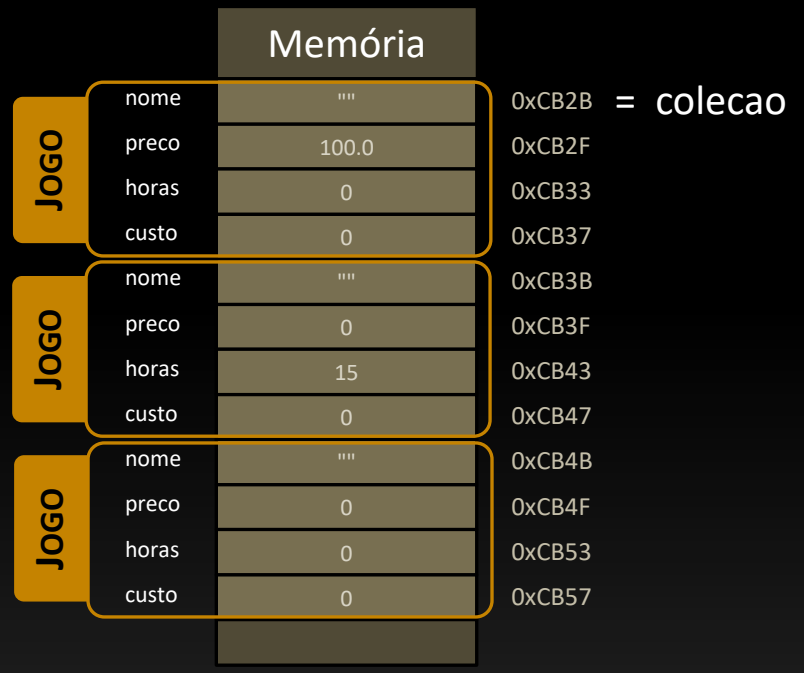
- A função trabalha com dois objetos

Vetor de Objetos

- Podemos criar vetores de objetos
  - Da mesma forma que criamos vetores de tipos primitivos

Requer a existência de um construtor padrão.

```
Jogo colecao[3];

colecao[0].atualizar(100.0f);
colecao[1].jogar(15);
colecao[2].exibir();
```



Vetor de Objetos

- É possível também inicializar os objetos com valores
  - Usando um dos construtores disponíveis

```
Jogo colecao[3] =
{
   Jogo("Gears", 90.0f),
   Jogo("Doom")
};
```

Requer um **construtor padrão** para elementos não inicializados.

Vetor de Objetos

```cpp
#include <iostream>
#include "Jogo.h"
using namespace std;

const int MAX = 3;

int main()
{
    // vetor de objetos inicializados
    Jogo colecao[MAX] =
    {
         Jogo("Gears", 90.0f, 30),
         Jogo("Doom", 60.0f, 120),
         Jogo("Halo", 80.0f, 40)
    };

    cout << "Coleção de Jogos:\n";
    for (int i = 0; i < MAX; i++)
        colecao[i].exibir();

    // aponta para primeiro elemento
    const Jogo * top = &colecao[0];

    // compara com todos os elementos
    for (int i = 1; i < MAX; i++)
        top = &top->comparar(colecao[i]);

    // top aponta para o mais jogado
    cout << "\nJogo mais jogado:\n";
    top->exibir();
}
```

» Colecao.cpp

Vetor de Objetos

- Saída do programa:

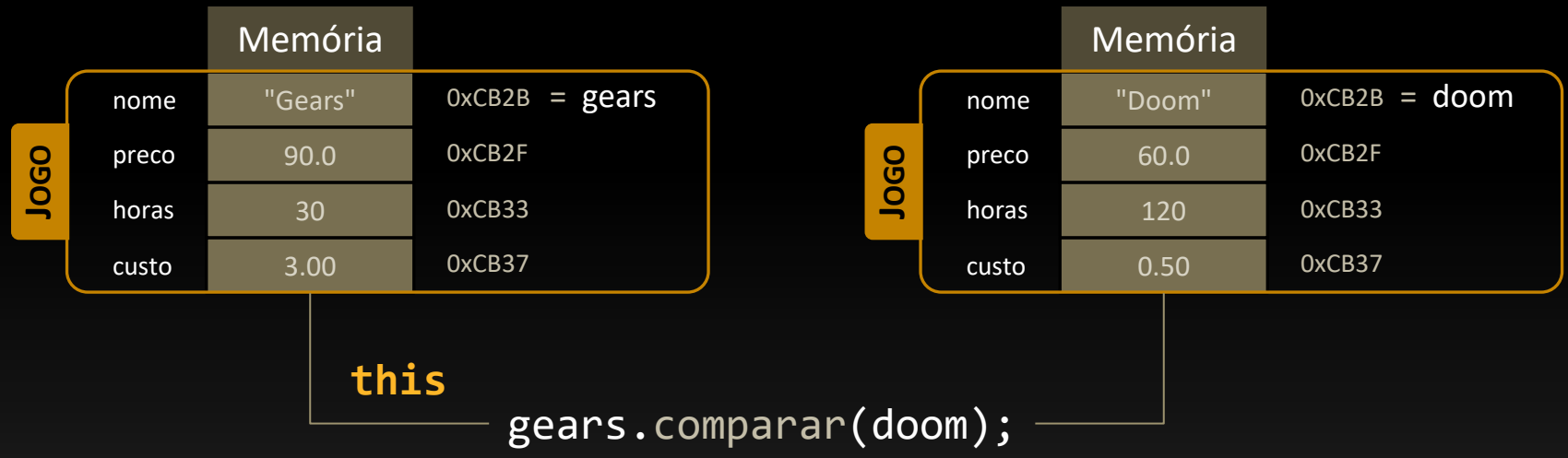
```
Coleção de Jogos:
Gears R$90.00 30h = R$3.00/h
Doom R$60.00 120h = R$0.50/h
Halo R$80.00 40h = R$2.00/h

Jogo mais jogado:
Doom R$60.00 120h = R$0.50/h
```

Comparar objetos nos permitiu conhecer o **ponteiro this**.

  - Poderíamos ter adota outras estratégias
    - Criar um método inline para retornar horas
    - Fazer um laço para obter e comparar o tempo de jogo

Resumo

- **Métodos** acessam atributos
  - De objetos passados por parâmetro
  - Do objeto usado para chamar o método



Resumo

- Não há diferença entre vetores:
  - De tipos primitivos
  - De registros
  - De objetos

Contanto que exista um construtor padrão para o objeto.

```
int vet[5];
Jogador time[22];
Jogo colecao[3];

colecao[0].atualizar(100.0f);
colecao[1].jogar(15);
colecao[2].exibir();
```

```
// construtor padrão
Jogo::Jogo()
{
    nome  = "";
    preco = 0;
    horas = 0;
    custo = 0;
}
```