# Classes em C++

## Mover Construtor em C++

Introdução

O mover construtor em C++ é utilizado para transferir os recursos de um objeto para outro, em vez de fazer uma cópia. Ele é essencial para otimizar o desempenho, especialmente quando se trabalha com recursos dinâmicos ou objetos grandes, minimizando cópias desnecessárias.

1. Definição e Sintaxe

- Definição: Um mover construtor é um construtor que transfere os recursos de um objeto para um novo objeto, deixando o objeto original em um estado válido, mas indefinido.
- Sintaxe:

```
class NomeClasse {
public:
    NomeClasse(NomeClasse&& outro) noexcept;  // Declaração do mover construtor
};
```

2. Mover Construtor Implicitamente Definido

- Definição: Se nenhuma forma de mover construtor for fornecida pela classe, o compilador gera automaticamente um mover construtor padrão que realiza a movimentação dos membros.
- Exemplo:

```
class Exemplo {
```

```
public:
    int* ptr;

    Exemplo() : ptr(new int(42)) {}

    // Mover construtor implicitamente definido
};

int main() {
    Exemplo obj1;
    Exemplo obj2 = std::move(obj1);  // Chama o mover construtor
    return 0;
}
```

3. Mover Construtor Explicitamente Definido

- Definição: Um mover construtor pode ser explicitamente definido pelo programador para realizar uma movimentação personalizada dos recursos.
- Exemplo:

```
class Exemplo {
public:
    int* ptr;

    Exemplo(int valor) : ptr(new int(valor)) {}
```

```
    // Mover construtor explicitamente definido
    Exemplo(Exemplo&& outro) noexcept : ptr(outro.ptr) {
        outro.ptr = nullptr;  // Deixa o objeto original em um estado válido, mas indefinido
    }

    ~Exemplo() {
        delete ptr;
    }
};

int main() {
    Exemplo obj1(42);
    Exemplo obj2 = std::move(obj1);  // Chama o mover construtor
    return 0;
}
```

4. Mover Construtores e Herança

- Definição: Quando se utiliza herança, é importante garantir que os mover construtores das classes base e derivadas funcionem corretamente.
- Exemplo:

```
class Base {
public:
    int* ptrBase;
```

```
    Base(int valor) : ptrBase(new int(valor)) {}

    // Mover construtor da base
    Base(Base&& outro) noexcept : ptrBase(outro.ptrBase) {
        outro.ptrBase = nullptr;
    }

    ~Base() {
        delete ptrBase;
    }
};

class Derivada : public Base {
public:
    int* ptrDerivada;

        Derivada(int valorBase, int valorDerivada) : Base(valorBase), ptrDerivada(new int(valorDerivada)) {}

    // Mover construtor da derivada
    Derivada(Derivada&& outro) noexcept : Base(std::move(outro)), ptrDerivada(outro.ptrDerivada) {
        outro.ptrDerivada = nullptr;
    }

    ~Derivada() {
```

```
        delete ptrDerivada;
    }
};

int main() {
    Derivada obj1(1, 2);
    Derivada obj2 = std::move(obj1);  // Chama o mover construtor
    return 0;
}
```

5. Mover Construtor `default`

- Definição: Um mover construtor pode ser explicitamente declarado como `default` para indicar que o compilador deve gerar a implementação padrão.
- Exemplo:

```
class Exemplo {
public:
    int* ptr;

    Exemplo(int v) : ptr(new int(v)) {}

    Exemplo(Exemplo&& outro) noexcept = default;  // Solicita ao compilador que gere o mover construtor padrão
};
```

```
int main() {
    Exemplo obj1(42);
    Exemplo obj2 = std::move(obj1);  // Chama o mover construtor padrão
    return 0;
}
```

6. Desabilitando o Mover Construtor

- Definição: O mover construtor pode ser desabilitado explicitamente para impedir a movimentação de objetos.
- Exemplo:

```
class Exemplo {
public:
    Exemplo(int valor) {
        // Construtor
    }

    Exemplo(Exemplo&& outro) = delete;  // Desabilita o mover construtor
};

int main() {
    Exemplo obj1(42);
    // Exemplo obj2 = std::move(obj1);  // Erro: mover construtor está desabilitado
    return 0;
}
```

**Classes em C++**

Dicas de Boas Práticas

- Uso de `noexcept`: Sempre que possível, marque o mover construtor com `noexcept` para permitir otimizações pelo compilador.
- Verificação de Auto-movimentação: Embora auto-movimentação não seja comum, é uma boa prática verificar e proteger contra isso, se necessário.
- Estado Válido: Garanta que o objeto original esteja em um estado válido após a movimentação, mesmo que seja um estado indefinido.

Esta seção abrange os conceitos sobre o mover construtor em C++. Para mais detalhes, consulte a documentação oficial: https://en.cppreference.com/w/cpp/language/move_constructor