# Classes em C++

## Construtor de Cópia em C++

Introdução

O construtor de cópia em C++ é utilizado para criar um novo objeto como uma cópia de um objeto existente. Ele é essencial para garantir que a cópia de objetos seja realizada corretamente, especialmente quando o objeto contém ponteiros ou outros recursos que requerem cópia profunda.

1. Definição e Sintaxe

- Definição: Um construtor de cópia é um construtor que inicializa um objeto usando outro objeto do mesmo tipo.
- Sintaxe:

```
class NomeClasse {
public:
    NomeClasse(const NomeClasse& outro);  // Declaração do construtor de cópia
};
```

2. Construtor de Cópia Implicitamente Definido

- Definição: Se nenhum construtor de cópia for explicitamente definido, o compilador gera automaticamente um construtor de cópia padrão que realiza a cópia membro a membro.
- Exemplo:

```
class Exemplo {
```

```
public:
    int valor;
};

int main() {
    Exemplo obj1;
    obj1.valor = 42;
    Exemplo obj2 = obj1;  // Construtor de cópia implicitamente definido
    return 0;
}
```

3. Construtor de Cópia Explicitamente Definido

- Definição: Um construtor de cópia pode ser explicitamente definido pelo programador para realizar uma cópia profunda ou outras operações especiais durante a cópia.
- Exemplo:

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

    Exemplo(int valor) {
        ptr = new int(valor);
    }

    // Construtor de cópia explicitamente definido
```

```
    Exemplo(const Exemplo& outro) {
        ptr = new int(*outro.ptr);  // Cópia profunda
    }

    ~Exemplo() {
        delete ptr;
    }
};

int main() {
    Exemplo obj1(42);
    Exemplo obj2 = obj1;  // Chama o construtor de cópia
    return 0;
}
```

4. Construtores de Cópia e Herança

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

```
class Base {
public:
    int valorBase;

    Base(int valor) : valorBase(valor) {}
```

```
    Base(const Base& outro) : valorBase(outro.valorBase) {}  // Construtor de cópia da base
};

class Derivada : public Base {
public:
    int valorDerivado;

    Derivada(int valorBase, int valorDerivado) : Base(valorBase), valorDerivado(valorDerivado) {}

        Derivada(const Derivada& outro) : Base(outro), valorDerivado(outro.valorDerivado) {}  // Construtor de cópia da derivada
};

int main() {
    Derivada obj1(1, 2);
    Derivada obj2 = obj1;  // Chama o construtor de cópia
    return 0;
}
```

5. Construtor de Cópia `default`

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

```
class Exemplo {
public:
    int valor;

    Exemplo(int v) : valor(v) {}

     Exemplo(const Exemplo& outro) = default;  // Solicita ao compilador que gere o construtor de cópia padrão
};

int main() {
    Exemplo obj1(42);
    Exemplo obj2 = obj1;  // Chama o construtor de cópia padrão
    return 0;
}
```

6. Desabilitando o Construtor de Cópia

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

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

```
    }

    Exemplo(const Exemplo& outro) = delete;  // Desabilita o construtor de cópia
};

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

Dicas de Boas Práticas

- Cópia Profunda vs. Cópia Superficial: Use cópia profunda para evitar problemas com ponteiros e recursos dinâmicos.
- Desempenho: Avalie o impacto de desempenho ao definir construtores de cópia, especialmente em classes com muitos membros ou recursos dinâmicos.
- Uso de `default`: Use `default` para indicar claramente que a implementação padrão do construtor de cópia deve ser gerada pelo compilador.

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