Classes em C++

Especificador `override` em C++

Introdução

O especificador `override` em C++ é usado para indicar que uma função de membro está substituindo uma função virtual de uma classe base. Ele ajuda a evitar erros acidentais ao garantir que a função realmente substitui uma função virtual da classe base.

1. Definição e Sintaxe

- Definição: O especificador `override` é adicionado à declaração de uma função virtual em uma classe derivada para indicar que a função está substituindo uma função virtual da classe base.
- Sintaxe:

```
class Base {
public:
    virtual void funcaoVirtual();
};

class Derivada : public Base {
public:
    void funcaoVirtual() override;  // Substitui a função da classe base
};
```

2. Exemplo de Uso do `override`

- Exemplo: O uso do `override` ajuda a garantir que a função na classe derivada está realmente substituindo uma função na classe base.

```
#include <iostream>

class Base {
public:
    virtual void mostrar() {
        std::cout << "Classe Base" << std::endl;
    }
};

class Derivada : public Base {
public:
    void mostrar() override {
        std::cout << "Classe Derivada" << std::endl;
    }
};

int main() {
    Base* b = new Derivada();
    b->mostrar();  // Chamará Derivada::mostrar
    delete b;
    return 0;
}
```

3. Erro de Compilação sem `override`

- Exemplo: Sem o `override`, é possível que a função derivada não substitua a função base devido a um erro de digitação ou mudança na assinatura da função. Isso não gera um erro de compilação.

```
class Base {
public:
    virtual void mostrar() {
        std::cout << "Classe Base" << std::endl;
    }
};

class Derivada : public Base {
public:
    void mostrar(int x) {  // Erro: não substitui a função base
        std::cout << "Classe Derivada" << std::endl;
    }
};
```

4. Vantagens do Uso de `override`

- Verificação pelo Compilador: O compilador verifica se a função está realmente substituindo uma função virtual da classe base.
- Evita Erros: Ajuda a evitar erros acidentais, como erros de digitação ou mudanças na assinatura da função, que podem resultar em substituições incorretas.

- Clareza do Código: Torna o código mais claro e fácil de entender, indicando explicitamente que uma função está substituindo outra função.

5. Exemplo Completo

- Exemplo:

```
#include <iostream>

class Base {
public:
    virtual void desenhar() {
        std::cout << "Desenhar base" << std::endl;
    }
};

class Circulo : public Base {
public:
    void desenhar() override {
        std::cout << "Desenhar círculo" << std::endl;
    }
};

class Quadrado : public Base {
public:
    void desenhar() override {
```

```
        std::cout << "Desenhar quadrado" << std::endl;
    }
};

int main() {
    Base* formas[] = {new Circulo(), new Quadrado()};
    for (Base* forma : formas) {
        forma->desenhar();
    }
    for (Base* forma : formas) {
        delete forma;
    }
    return 0;
}
```

Dicas de Boas Práticas

- Sempre Usar `override`: Sempre use `override` quando substituir funções virtuais para garantir a verificação pelo compilador.
- Manutenção de Código: Facilita a manutenção do código ao deixar claro quais funções estão substituindo funções de classes base.
- Consistência: Mantenha a consistência no uso do `override` para aumentar a legibilidade e robustez do código.

Esta seção abrange os conceitos sobre o especificador `override` em C++. Para mais detalhes,

consulte a documentação oficial: https://en.cppreference.com/w/cpp/language/override