Como o Polimorfismo se Apresenta em C++

O polimorfismo é um dos pilares da Programação Orientada a Objetos (POO) e permite que objetos de diferentes classes sejam tratados como objetos de uma classe base, enquanto ainda mantêm suas implementações específicas. Em C++, o polimorfismo pode se apresentar de duas formas principais:

---

1. Polimorfismo em Tempo de Compilação (Estático)

O polimorfismo estático ocorre quando a função a ser executada é determinada em tempo de compilação. Ele é implementado principalmente por meio de:

1.1 Sobrecarga de Funções (Function Overloading)

- Várias funções com o mesmo nome, mas com parâmetros diferentes, podem coexistir na mesma classe.
- O compilador decide qual função chamar com base nos argumentos passados.

Exemplo:

```
#include <iostream>

class Calculadora {
public:
    int somar(int a, int b) {
        return a + b;
    }

    double somar(double a, double b) {
        return a + b;
    }
};

int main() {
    Calculadora calc;
    std::cout << calc.somar(2, 3) << std::endl;       // Chama int somar(int, int)
    std::cout << calc.somar(2.5, 3.5) << std::endl;   // Chama double
somar(double, double)
    return 0;
}
```

1.2 Sobrecarga de Operadores (Operator Overloading)

- Permite definir comportamentos personalizados para operadores (como +, -, ==, etc.) em classes.

Exemplo:

```
#include <iostream>

class Vetor {
```

```
public:
    int x, y;

    Vetor(int x, int y) : x(x), y(y) {}

    Vetor operator+(const Vetor& outro) {
        return Vetor(x + outro.x, y + outro.y);
    }
};

int main() {
    Vetor v1(1, 2);
    Vetor v2(3, 4);
    Vetor resultado = v1 + v2; // Usa o operador sobrecarregado
    std::cout << "Resultado: (" << resultado.x << ", " << resultado.y << ")" <<
std::endl;
    return 0;
}
```

---

2. Polimorfismo em Tempo de Execução (Dinâmico)

O polimorfismo dinâmico ocorre quando a função a ser executada é determinada em tempo de execução, com base no tipo real do objeto. Ele é implementado principalmente por meio de:

2.1 Funções Virtuais

- Funções declaradas como `virtual` em uma classe base podem ser sobrescritas (override) em classes derivadas.
- O compilador usa uma **tabela de funções virtuais (vtable)** para determinar qual função chamar em tempo de execução.

Exemplo:

```
#include <iostream>

class Animal {
public:
    virtual void fazerSom() {
        std::cout << "Som genérico de animal" << std::endl;
    }
};

class Cachorro : public Animal {
public:
    void fazerSom() override {
        std::cout << "Au Au!" << std::endl;
    }
};

int main() {
```

```
    Animal* animal = new Cachorro();
    animal->fazerSom(); // Saída: Au Au! (chama a função de Cachorro)
    delete animal;
    return 0;
}
```

2.2 Funções Virtuais Puras e Classes Abstratas

- Funções virtuais puras são declaradas com `= 0` e não possuem implementação na classe base.
- Classes com funções virtuais puras são chamadas de **classes abstratas** e não podem ser instanciadas diretamente.
- Classes derivadas devem implementar todas as funções virtuais puras para se tornarem concretas.

Exemplo:

```cpp
#include <iostream>

class Forma {
public:
    virtual void desenhar() = 0; // Função virtual pura
};

class Circulo : public Forma {
public:
    void desenhar() override {
        std::cout << "Desenhando um círculo" << std::endl;
    }
};

int main() {
    Forma* forma = new Circulo();
    forma->desenhar(); // Saída: Desenhando um círculo
    delete forma;
    return 0;
}
```

---

3. Comparação entre Polimorfismo Estático e Dinâmico

Característica	Polimorfismo Estático	Polimorfismo Dinâmico
Resolução da função	Em tempo de compilação.	Em tempo de execução.
Mecanismo	Sobrecarga de funções e operadores.	Funções virtuais e tabela de funções (vtable).
Flexibilidade	Menos flexível, pois depende dos tipos conhecidos em tempo de compilação.	Mais flexível, pois permite decisões em tempo de execução.

Característica	Polimorfismo Estático	Polimorfismo Dinâmico
Desempenho	Mais rápido, pois não há overhead de vtable.	Mais lento, devido ao overhead de vtable.

4. Quando Usar Cada Tipo de Polimorfismo

- **Polimorfismo Estático**:
  - Use quando o comportamento pode ser determinado em tempo de compilação.
  - Ideal para operações simples e de alto desempenho.
- **Polimorfismo Dinâmico**:
  - Use quando o comportamento depende do tipo real do objeto em tempo de execução.
  - Ideal para sistemas extensíveis e com hierarquias complexas de classes.

---

5. Conclusão

O polimorfismo em C++ é uma ferramenta poderosa que permite escrever código flexível e reutilizável. Ele pode ser implementado de duas formas:

1. **Polimorfismo Estático**: Resolvido em tempo de compilação, usando sobrecarga de funções e operadores.
2. **Polimorfismo Dinâmico**: Resolvido em tempo de execução, usando funções virtuais e classes abstratas.

Ambas as formas têm seus usos e benefícios, e a escolha entre elas depende das necessidades do projeto. Compreender esses conceitos é essencial para dominar a POO em C++.