Classes em C++

Destrutor em C++

Introdução

Destrutores em C++ são funções especiais que são chamadas automaticamente quando um objeto sai de escopo ou é explicitamente deletado. Eles são usados para liberar recursos adquiridos pela classe, como memória dinâmica, arquivos e conexões de rede.

1. Definição e Sintaxe

- Definição: Um destrutor é uma função membro especial de uma classe que é executada quando um objeto da classe é destruído.
- Sintaxe:

```
class NomeClasse {
public:
    ~NomeClasse();  // Declaração do destrutor
};
```

2. Destrutor Implicitamente Definido

- Definição: Se nenhum destrutor for definido explicitamente, o compilador gera automaticamente um destrutor padrão.
- Exemplo:

```
class Exemplo {
```

```
public:
    int* ptr;

    Exemplo() {
        ptr = new int;
    }

    // Destrutor implicitamente definido
};

int main() {
    Exemplo obj;
    return 0;  // O destrutor é chamado automaticamente
}
```

3. Destrutor Explicitamente Definido

- Definição: Um destrutor pode ser explicitamente definido pelo programador para liberar recursos adquiridos pela classe.
- Exemplo:

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

    Exemplo() {
```

```
        ptr = new int;
    }

    ~Exemplo() {  // Destrutor explicitamente definido
        delete ptr;
    }
};

int main() {
    Exemplo obj;
    return 0;  // O destrutor é chamado automaticamente
}
```

4. Destrutores e Herança

- Definição: Em uma hierarquia de classes, os destrutores das classes derivadas e da classe base são chamados automaticamente na ordem correta.
- Exemplo:

```
class Base {
public:
    Base() {
        std::cout << "Construtor da Base" << std::endl;
    }

    virtual ~Base() {  // Destrutor virtual
```

```
        std::cout << "Destrutor da Base" << std::endl;
    }
};

class Derivada : public Base {
public:
    Derivada() {
        std::cout << "Construtor da Derivada" << std::endl;
    }

    ~Derivada() {
        std::cout << "Destrutor da Derivada" << std::endl;
    }
};

int main() {
    Base* obj = new Derivada();
    delete obj;  // Chama ~Derivada e depois ~Base
    return 0;
}
```

5. Destrutor `default`

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

Classes em C++

- Exemplo:

```
class Exemplo {
public:
    ~Exemplo() = default;  // Solicita ao compilador que gere o destrutor padrão
};

int main() {
    Exemplo obj;
    return 0;
}
```

6. Desabilitando o Destrutor

- Definição: O destrutor pode ser desabilitado explicitamente para impedir a destruição de objetos.
- Exemplo:

```
class Exemplo {
public:
    ~Exemplo() = delete;  // Desabilita o destrutor

    void liberar() {
        // Função customizada para liberar recursos
    }
};

int main() {
```

```
    // Exemplo obj;  // Erro: destrutor está desabilitado
    Exemplo* obj = new Exemplo();
    obj->liberar();
    // delete obj;  // Erro: destrutor está desabilitado
    return 0;
}
```

Dicas de Boas Práticas

- Liberação de Recursos: Sempre libere todos os recursos adquiridos pela classe no destrutor para evitar vazamentos de memória e outros problemas.
- Destrutores Virtuais: Declare destrutores como virtuais em classes base que possuem outras funções virtuais para garantir que os destrutores das classes derivadas sejam chamados corretamente.
- Uso de `default`: Use `default` para indicar claramente que a implementação padrão do destrutor deve ser gerada pelo compilador.

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