Templates em C++

Template de Classe em C++

Introdução

Templates de classe em C++ permitem a criação de classes genéricas que podem operar com qualquer tipo de dado. Isso é extremamente útil para escrever código reutilizável e flexível.

1. Definição e Sintaxe

- Definição: Um template de classe é um modelo de classe que pode ser instanciado com diferentes tipos de dados.
- Sintaxe:

```
template <typename T>
class NomeClasse {
    // Definição da classe
};
```

2. Exemplo Básico de Template de Classe

- Definição: Um template de classe básico que armazena um valor do tipo genérico T.
- Exemplo:

```
template <typename T>
class Caixa {
private:
```

```
    T valor;

public:
    Caixa(T v) : valor(v) {}
    T getValor() const { return valor; }
};

int main() {
    Caixa<int> caixaInt(10);
    Caixa<double> caixaDouble(10.5);
    std::cout << caixaInt.getValor() << std::endl; // Uso do template de classe com int
    std::cout << caixaDouble.getValor() << std::endl; // Uso do template de classe com double
    return 0;
}
```

3. Métodos de Template de Classe

- Definição: Métodos de template de classe podem ser definidos fora da definição da classe.
- Exemplo:

```
template <typename T>
class Caixa {
private:
    T valor;

public:
```

```
    Caixa(T v);
    T getValor() const;
};

template <typename T>
Caixa<T>::Caixa(T v) : valor(v) {}

template <typename T>
T Caixa<T>::getValor() const {
    return valor;
}

int main() {
    Caixa<int> caixaInt(10);
    Caixa<double> caixaDouble(10.5);
    std::cout << caixaInt.getValor() << std::endl; // Uso do template de classe com int
    std::cout << caixaDouble.getValor() << std::endl; // Uso do template de classe com double
    return 0;
}
```

4. Especificação de Template de Classe

- Definição: Templates de classe podem ser especializados para tipos específicos.
- Exemplo:

```
template <typename T>
```

```
class Exemplo {
public:
    void funcao() {
        std::cout << "Template genérico" << std::endl;
    }
};

template <>
class Exemplo<int> {
public:
    void funcao() {
        std::cout << "Template especializado para int" << std::endl;
    }
};

int main() {
    Exemplo<double> obj1;
    Exemplo<int> obj2;
    obj1.funcao(); // Chamará a versão genérica
    obj2.funcao(); // Chamará a versão especializada para int
    return 0;
}
```

5. Template de Classe Parcialmente Especializado

Templates em C++

- Definição: É possível especializar parcialmente um template de classe.

- Exemplo:

```
template <typename T, typename U>
class Par {
public:
    T primeiro;
    U segundo;

    Par(T p, U s) : primeiro(p), segundo(s) {}
};

template <typename T>
class Par<T, int> {
public:
    T primeiro;
    int segundo;

    Par(T p, int s) : primeiro(p), segundo(s) {}
    void mostrar() {
        std::cout << primeiro << ", " << segundo << std::endl;
    }
};

int main() {
    Par<double, int> par(3.14, 42);
```

```
    par.mostrar(); // Uso do template de classe parcialmente especializado
    return 0;
}
```

6. Template de Classe com Múltiplos Parâmetros

- Definição: Templates de classe podem ter múltiplos parâmetros de template.
- Exemplo:

```
template <typename T, typename U>
class Par {
public:
    T primeiro;
    U segundo;

    Par(T p, U s) : primeiro(p), segundo(s) {}
    void mostrar() {
        std::cout << primeiro << ", " << segundo << std::endl;
    }
};

int main() {
    Par<int, double> par(10, 3.14);
    par.mostrar(); // Uso do template de classe com múltiplos parâmetros
    return 0;
}
```

Templates em C++

7. Template de Classe com Parâmetros Não-Tipo

- Definição: Templates de classe podem ter parâmetros de template que não são tipos.
- Exemplo:

```
template <typename T, int N>
class Array {
private:
    T arr[N];

public:
    T& operator[](int index) {
        return arr[index];
    }
};

int main() {
    Array<int, 10> arrayInt;
    arrayInt[0] = 1;
    std::cout << arrayInt[0] << std::endl; // Uso do template de classe com parâmetro não-tipo
    return 0;
}
```

Dicas de Boas Práticas

Templates em C++

- Reutilização de Código: Use templates de classe para criar classes genéricas e reutilizáveis.
- Especialização: Especialize templates de classe quando necessário para lidar com tipos específicos.
- Clareza e Manutenção: Mantenha os templates de classe claros e bem documentados para facilitar a manutenção do código.

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