Templates em C++

Introdução

Templates em C++ permitem a criação de funções e classes genéricas, que podem operar com qualquer tipo de dados. Eles são uma característica poderosa da linguagem, permitindo reutilização de código e suporte a programação genérica.

1. Definição e Sintaxe

- Definição: Templates são modelos que podem ser utilizados para criar funções e classes genéricas.
- Sintaxe:

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

template <typename T>
T nomeFuncao(T argumento) {
    // Definição da função
}
```

2. Templates de Função

Templates em C++

- Definição: Templates de função permitem criar funções que podem operar com qualquer tipo de dado.

- Exemplo:

```
template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}

int main() {
    int x = 10, y = 20;
    double a = 10.5, b = 20.5;
    std::cout << max(x, y) << std::endl; // Uso do template de função com int
    std::cout << max(a, b) << std::endl; // Uso do template de função com double
    return 0;
}
```

3. Templates de Classe

- Definição: Templates de classe permitem criar classes que podem operar com qualquer tipo de dado.

- Exemplo:

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

```
    T valor;

public:
    Caixa(T v) : valor(v) {}
    T getValor() { 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 Templates

- Definição: É possível especificar diferentes implementações para tipos específicos usando templates especializados.
- 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. Templates de Variáveis

- Definição: Introduzido no C++14, permite a criação de variáveis template.
- Exemplo:

```
template <typename T>
constexpr T pi = T(3.1415926535897932385);
```

```
int main() {
    std::cout << pi<double> << std::endl; // Uso do template de variável com double
    std::cout << pi<float> << std::endl;  // Uso do template de variável com float
    return 0;
}
```

6. Template Template Parameters

- Definição: Permite que templates aceitem outros templates como parâmetros.
- Exemplo:

```
template <template <typename> class Container, typename T>
class CaixaDeCaixas {
private:
    Container<T> caixa;

public:
    CaixaDeCaixas(T v) : caixa(v) {}
    T getValor() { return caixa.getValor(); }
};

int main() {
    CaixaDeCaixas<Caixa, int> caixaInt(10);
    std::cout << caixaInt.getValor() << std::endl; // Uso de template template parameter
    return 0;
```

```
}
```

7. Aliases de Templates

- Definição: Introduzido no C++11, permite a criação de aliases para templates.
- Exemplo:

```
template <typename T>
using Vec = std::vector<T>;

int main() {
    Vec<int> vetorInt;
    vetorInt.push_back(10);
    std::cout << vetorInt[0] << std::endl; // Uso do alias de template
    return 0;
}
```

Dicas de Boas Práticas

- Reutilização de Código: Use templates para criar código genérico que pode ser reutilizado com diferentes tipos de dados.
- Especialização: Especialize templates quando for necessário tratar tipos específicos de maneira diferente.
- Clareza e Simplicidade: Mantenha os templates claros e simples para facilitar a leitura e manutenção do código.

Templates em C++

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