Templates em C++

Bloco Try em C++

Introdução

O bloco try em C++ é utilizado para detectar e tratar exceções. Ele permite que você execute um conjunto de instruções que podem lançar exceções e, em seguida, trate essas exceções usando cláusulas catch.

1. Definição e Sintaxe

- Definição: O bloco try é usado para envolver código que pode lançar exceções. As exceções são capturadas e tratadas em blocos catch subsequentes.
- Sintaxe:

```
try {
    // Código que pode lançar uma exceção
} catch (const std::exception& e) {
    // Código para tratar a exceção
}
```

2. Uso Básico do Bloco try

- Definição: Envolva o código que pode lançar exceções em um bloco try e capture as exceções com blocos catch.
- Exemplo:

```
int main() {
    try {
        throw std::runtime_error("Erro ocorrido");
    } catch (const std::runtime_error& e) {
        std::cout << "Exceção capturada: " << e.what() << std::endl;
    }
    return 0;
}
```

3. Múltiplos Blocos catch

- Definição: Você pode usar múltiplos blocos catch para capturar diferentes tipos de exceções.
- Exemplo:

```
int main() {
    try {
        throw std::runtime_error("Erro de tempo de execução");
    } catch (const std::logic_error& e) {
        std::cout << "Exceção lógica capturada: " << e.what() << std::endl;
    } catch (const std::runtime_error& e) {
        std::cout << "Exceção de tempo de execução capturada: " << e.what() << std::endl;
    }
    return 0;
}
```

4. Bloco catch Genérico

- Definição: Você pode usar um bloco catch genérico para capturar qualquer exceção.
- Exemplo:

```
int main() {
    try {
        throw std::runtime_error("Erro ocorrido");
    } catch (const std::exception& e) {
        std::cout << "Exceção capturada: " << e.what() << std::endl;
    } catch (...) {
        std::cout << "Exceção desconhecida capturada" << std::endl;
    }
    return 0;
}
```

5. Re-Lançando Exceções

- Definição: Você pode re-lançar uma exceção capturada usando a palavra-chave throw dentro de um bloco catch.
- Exemplo:

```
int main() {
    try {
        try {
            throw std::runtime_error("Erro original");
        } catch (const std::runtime_error& e) {
            std::cout << "Tratando e re-lançando: " << e.what() << std::endl;
```

```
            throw;  // Re-lança a exceção
        }
    } catch (const std::runtime_error& e) {
        std::cout << "Exceção re-lançada capturada: " << e.what() << std::endl;
    }
    return 0;
}
```

6. Bloco try em Funções

- Definição: Você pode usar blocos try dentro de funções para tratar exceções locais.
- Exemplo:

```
void funcao() {
    try {
        throw std::runtime_error("Erro na função");
    } catch (const std::runtime_error& e) {
        std::cout << "Exceção capturada na função: " << e.what() << std::endl;
    }
}

int main() {
    funcao();
    return 0;
}
```

7. Bloco try em Inicializadores de Construtores

- Definição: Você pode usar blocos try em inicializadores de construtores para capturar exceções lançadas durante a inicialização de membros.
- Exemplo:

```
class Exemplo {
public:
    Exemplo() try : membro(new int[10]) {
        throw std::runtime_error("Erro no construtor");
    } catch (const std::exception& e) {
        std::cout << "Exceção capturada no construtor: " << e.what() << std::endl;
    }

private:
    int* membro;
};

int main() {
    try {
        Exemplo e;
    } catch (const std::exception& e) {
        std::cout << "Exceção capturada no main: " << e.what() << std::endl;
    }
    return 0;
}
```

8. Bloco try com Recursos de noexcept

- Definição: Use a especificação noexcept para funções que não lançam exceções. A combinação com blocos try ajuda a garantir segurança e eficiência.
- Exemplo:

```
void funcao() noexcept {
    // Função que não lança exceções
}

int main() {
    try {
        funcao();
    } catch (...) {
        std::cout << "Isso nunca será impresso" << std::endl;
    }
    return 0;
}
```

Dicas de Boas Práticas

- Uso Apropriado de Blocos try: Utilize blocos try para envolver código que pode lançar exceções, garantindo um tratamento adequado de erros.
- Clareza e Manutenção: Mantenha o código de tratamento de exceções claro e bem documentado para facilitar a leitura e a manutenção do código.

- Evite Blocos catch Vazios: Certifique-se de tratar adequadamente as exceções capturadas e não simplesmente ignorá-las.

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