Templates em C++

Exceções em C++

Introdução

Exceções em C++ são um mecanismo para lidar com erros e condições excepcionais de forma estruturada. Elas permitem que você separe o código de tratamento de erros do código principal, melhorando a legibilidade e a manutenibilidade do programa.

1. Definição e Sintaxe

- Definição: Exceções são usadas para indicar e tratar condições de erro ou situações excepcionais em um programa.
- Sintaxe:

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

2. Lançando Exceções

- Definição: O lançamento de exceções é feito usando a palavra-chave throw.
- Exemplo:

```
void funcao() {
```

```
    throw std::runtime_error("Erro ocorrido");
}

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

3. Capturando Exceções

- Definição: A captura de exceções é feita usando a palavra-chave 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;
}
```

4. Exceções Padrão

- Definição: A biblioteca padrão C++ inclui várias exceções padrão, como std::exception, std::runtime_error, std::logic_error, etc.
- Exemplo:

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

5. Definindo Exceções Personalizadas

- Definição: Você pode definir suas próprias exceções derivando da classe std::exception.
- Exemplo:

```
class MinhaExcecao : public std::exception {
public:
    const char* what() const noexcept override {
        return "Minha exceção personalizada";
    }
};
```

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

6. Propagação de Exceções

- Definição: Exceções podem ser propagadas para fora da função onde foram lançadas e capturadas em um nível superior da chamada de pilha.
- Exemplo:

```
void funcao() {
    throw std::runtime_error("Erro na função");
}

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

```
}
```

7. Multiplas Clausulas Catch

- Definição: Você pode usar múltiplas cláusulas catch para tratar diferentes tipos de exceções.
- Exemplo:

```
int main() {
    try {
        throw std::runtime_error("Erro ocorrido");
    } 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;
}
```

8. Cláusula noexcept

- Definição: A especificação noexcept é usada para indicar que uma função não lança exceções.
- Exemplo:

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

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

9. Cláusula throw

- Definição: A cláusula throw pode ser usada para especificar quais exceções uma função pode lançar. No entanto, seu uso é descontinuado no C++11 e posterior em favor de noexcept.
- Exemplo:

```
void funcao() throw(std::runtime_error) {
    throw std::runtime_error("Erro");
}

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

```
}
```

Dicas de Boas Práticas

- Uso Apropriado de Exceções: Utilize exceções para condições de erro e situações excepcionais, não para controle de fluxo regular.
- Clareza e Manutenção: Mantenha o código de tratamento de exceções claro e bem documentado para facilitar a leitura e a manutenção do código.
- Evite Swallowing de Exceções: Certifique-se de tratar adequadamente as exceções capturadas e não simplesmente ignorá-las.

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