PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Delano M. Beder & Prof. Renato Bueno

Tratamento de Exceções em C++

O tratamento de exceções permite capturar erros ocorridos durante a execução de um programa. Para tanto, um programa pode **'lançar'** e **'capturar'** exceções.

Para lançar uma exceção um objeto de uma classe (escolhida pelo programador) deve ser instanciado através do comando **throw (xxxx)**. No lugar do parâmetro 'xxxx' deve ser colocado um objeto (instância da classe escolhida pelo programador para representar a exceção). Esse objeto pode ser construído através da invocação do construtor da classe em que podem ser passados parâmetros de forma que no momento da **'captura'** da exceção se tenha dados sobre a mesma.

```
throw(DivZeroException(__FILE__, __LINE__));
```

Alternativamente, o parâmetro 'xxxx' pode ser um valor/variável de um tipo primitivo da linguagem (int, float, etc)

```
throw(10);
```

Ao lançar uma exceção o fluxo de execução do programa é desviado para o primeiro comando após o final do bloco **try { }**. Este comando deve ser um **catch(xxxx)**.
Do ponto de vista da linguagem C++, o par **try-catch** formam um único comando.

O comando **catch(xxxx)** é usado para capturar uma exceção, o parâmetro 'xxxx' especifica qual categoria de exceção será tratada neste bloco. Esta categoria é identificada pelo nome da classe (ou tipo) que foi usado no momento do lançamento da exceção com o comando **throw**.

```
try { // inicio do bloco try
   double valor = Calculadora::divide(a, b);
   // Os comandos abaixo não serão executados se exceção for lançada
   // no método divide(a,b).
   cout << "Divisão: " <<  valor << endl;
   if (a < 0 || b < 0) {
      throw(-1); // Lança uma exceção como um int
   }
} // final do bloco TRY
catch (DivZeroException e) {
  // pode ter ou não o objeto
  // apenas o tipo é obrigatório
  cout << "Tratador da exceção DivZeroException foi invocado ..." << endl;
  e.Msg();
}
```

(Ver exemplo completo em https://replit.com/@delanobeder/EH-01)

Caso uma exceção seja lançada e não haja um **catch** para tratá-la, o programa encerra de forma anormal, exibindo uma mensagem de erro.

No código acima, por exemplo, há somente um tratador para a exceção que lança um objeto da classe **DivZeroException**, assim, se for lançada uma exceção com um valor int, usando o comando **throw (-1);**

O programa irá encerrar com uma mensagem de erro semelhante a que segue:

```
terminate called after throwing an instance of 'int'
Aborted (core dumped)
```

Inserindo vários tratadores de exceção

É possível e muito comum, que um programa tenha vários tratadores de exceção **(catch)** em um bloco **try**.

Note que para cada tipo de exceção pode haver um tratador. Se houver mais de uma, apenas a primeira será executada.

No exemplo abaixo, no bloco **try** há 3 tratadores de exceções:
- quando objetos da classe **DivZeroException** são lançados (**throw**)
- quando objetos da classe **string** são lançados (**throw**)
- e, se ocorrer o lançamento de uma exceção não prevista pelos tratadores anteriores, essa exceção pode ser tratada pelo tratador genérico **catch (...)**.

```
try { // inicio do bloco try
    double valor = Calculadora::divide(a, b);
    // Os comandos abaixo não serão executados se DivZeroException for lançada.

    cout << "Divisão: " <<  valor << endl;
    valor = Calculadora::raiz(a);
    // Os comandos abaixo não serão executados se exceção for lançada.
    cout << "Raiz Quadrada de " << a << ": " << valor << endl;
    if (b < 0) {
        throw(b); // Lança uma exceção como um int
    }
} // final do bloco try
catch (DivZeroException e) {
  // pode ter ou não o objeto. apenas o tipo é obrigatório
  cout << "Tratador da exceção DivZeroException foi invocado ..." << endl;
  e.Msg();
}
catch (string S) {
 // pode ter ou não o objeto. apenas o tipo é obrigatório
 cout << "Tratador de exceção de string foi invocado ..." << endl;
 cout << S << endl;
}
catch (...) {
  cout << "Tratador de exceção Genérico !!" << endl;
}
```

(Ver exemplo completo em https://replit.com/@delanobeder/EH-02)

Exceções Padronizadas (Exemplo 03)

http://www.cplusplus.com/reference/exception/exception/

As exceções padronizadas permitem capturar algumas exceções geradas por comandos ou funções da biblioteca de C++. As mais comuns delas são: (1) **bad_alloc** que informa que faltou memória para executar o comando **new** e (2) **invalid_argument** que informa que os argumentos/parâmetros de um método são inválidos.

```
#include <iostream>
#include <exception>
using namespace std;

void alloc(int n) {
    if (n < 0) {
        throw invalid_argument("alloc: argumento deve ser positivo !!");
    }
    for (int i = 1; i <= n; i++) {
            new int[1234567890];
            cout << "Alocado (" << i << ") ..." << endl;
    }
}

int main() {
    try {
        int N;
        cout << "Valor de N: ";
        cin >> N;
        alloc(N);
    }
    catch (invalid_argument const ia) {
        cout << ia.what() << endl;
    }
    catch (bad_alloc const &) {
        cout << "Faltou Memoria..." << endl;
    }
}
```

(Ver exemplo completo em https://replit.com/@delanobeder/EH-03)

Leituras adicionais

- Exception Handling in C++
  https://www.geeksforgeeks.org/exception-handling-c/

- Throwing Exceptions in C++
  https://rollbar.com/blog/error-exceptions-in-c/#