IMD0030
LINGUAGEM DE PROGRAMAÇÃO I

Aula 30 – Tratamento de Exceções

INSTITUTO METRÓPOLE DIGITAL
UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Objetivos desta aula

- Aprender como detectar, tratar e lançar exceções em programas implementados na linguagem C++
- Para isto estudaremos:
  - O que são exceções
  - Como e porque tratar exceções em programas em C++
  - As diferentes bibliotecas relacionadas a exceções
- Ao final da aula espera-se que o aluno seja capaz de:
  - Compreender como tratar exceções lançadas por um programa em C++

Ocorreu um problema e seu PC precisa ser reiniciado. Estamos coletando algumas informações sobre o erro e, em seguida, reiniciaremos para você.

Se desejar saber mais, pesquise online mais tarde por este erro: CRITICAL_PROCESS_DIED

O que são exceções?

- Uma **exceção** indica que um **problema** ocorreu durante a **execução de um programa**
  - **Condição anormal** que faz com que o programa apresente erro e não execute conforme o esperado
- Exceções podem ser causadas principalmente por
  - erro a nível do sistema (falha em memória, permissões de arquivos, etc.)
  - erro do programador (operações inválidas, estouro de memória, etc.)

O que significa tratar exceções?

- O **tratamento de exceções** representa um mecanismo padronizado para **detectar** eventuais erros e como **reagir** uma vez que eles ocorram
  - Permitir o programa continuar sua execução
  - Notificar o usuário de que um erro ocorreu
  - Finalizar imediatamente a execução do programa
- Tratar exceções é importante para permitir o desenvolvimento de programas **robustos** e **tolerantes a falhas**
  - Programas capazes de lidar com eventuais erros que ocorram
  - Programas capazes de se recuperar de erros e/ou lhes dar um tratamento adequado

O que significa tratar exceções?

- Diz-se que o mecanismo de tratamento de exceções é feito para lidar com **erros síncronos**, gerados pela execução do programa
- Eventos ditos **assíncronos** (erros de disco, falhas na rede, etc.) não podem ser tratados com esse mecanismo
  - Podem ocorrer em paralelo à execução do programa
  - Ocorrem de forma independente do fluxo de controle do programa

Um exemplo

- Na Matemática, não é permitido realizar uma divisão por zero
- Na programação em C++
  - uma divisão entre números inteiros cujo denominador seja zero faz com que o programa seja encerrado imediatamente
  - uma divisão entre números de ponto flutuante retornam um valor inválido (`inf` ou `-inf`, infinito)
- Como tratar esse tipo de problema?

Um exemplo

Uma possibilidade

```
#include <iostream>
using std::cerr;
using std::cin;
using std::cout;

int main() {
   int numerador, denominador;
   cout << "Digite numerador e denominador: ";
   cin >> numerador >> denominador;

   if (denominador == 0) {
      cerr << "Operacao invalida: divisao por zero" << std::endl;
      return 1;
   } else {
      cout << "Resultado da divisao: " << (double) numerador / denominador << endl;
   }
   return 0;
}
```

Um exemplo

Uma possibilidade

- Detectar a condição que leva a um erro por meio de `if`/`else`
- Exibir uma mensagem de erro ao usuário utilizando `cerr` ou `cout` (biblioteca `<iostream>`)
- Tomar alguma atitude
  - Encerrar o programa (fazendo com que `main` retorne valor diferente de zero ou chamando a função `exit`)
  - Solicitar ao usuário que entre novamente com os valores

Qual a principal limitação dessa abordagem?

Replicação de código se há a possibilidade de o erro ocorrer em diferentes partes do programa, principalmente se ele tiver um tamanho considerável

Um exemplo

Uma alternativa

- **Detectar a condição** que leva a um erro
- **Lançar uma exceção** para indicar que se trata de um erro
- **Tratar a exceção** lançada para tomar algum tipo de ação
  - Exibir mensagem de erro para o usuário
  - Terminar o programa
  - Solicitar uma ação do usuário
  - Registrar o erro em um arquivo de *log*

Exceções

Em C++, **uma exceção é um objeto**

- Geralmente define-se uma **classe que representa um tipo de exceção**
  - C++ provê **classes padrão para exceções**, a partir das quais o programador pode criar suas próprias classes de exceção por meio de **herança**
- Lançar uma exceção significa **instanciar um objeto** da classe que representa tal exceção
- É possível ter diferentes **tipos de exceção**
  - Operação inválida (e.g., divisão por zero)
  - Erro em alocação dinâmica de memória
  - Acesso a posição inexistente em vetor
  - etc.

Classes padrão para exceções

C++ provê duas bibliotecas contendo classes padrão para exceções

- `<exception>`: define **apenas uma classe padrão** (`std::exception`) partir da qual o programador pode definir suas próprias classes de exceção
  - `std::exception` define um **método virtual** chamado `what`, que retorna uma mensagem (*string*) referente ao erro em questão a ser exibida para o usuário
  - O método `what` pode ser sobrescrito pela classe de exceção derivada para descrever de forma apropriada a exceção em questão
- `<stdexcept>`: define **diferentes classes de exceção** derivadas de `std::exception`
  - `std::runtime_error`: erro detectado em tempo de execução
  - `std::invalid_argument`: erro relacionado a argumento inválido recebido pelo programa
  - `std::out_of_range`: acesso a índice inexistente em um *container*

Classes padrão para exceções

Uma alternativa

```
#include <exception>
using std::exception;

class DivisaoPorZero : public exception {
   public:
      const char* what() {
         return "Operacao invalida: Divisao por zero";
      }
};
```

O método virtual `what` definido na classe `exception` (biblioteca `<exception>`) é sobrescrito pela classe derivada `DivisaoPorZero`

Classes padrão para exceções

Outra alternativa

```
#include <stdexcept>
using std::invalid_argument;

class DivisaoPorZero : public invalid_argument {
   public:
      DivisaoPorZero() : invalid_argument("Operacao invalida: Divisao por zero") {}
};
```

A classe `DivisaoPorZero` é derivada da classe `invalid_argument` (biblioteca `<stdexcept>`)

O construtor da classe `invalid_argument` recebe como parâmetro a mensagem de erro a ser exibida

Detectando e tratando exceções

C++ provê um bloco relacionado à **detecção e tratamento de exceções**, o `try`/`catch`

- O bloco `try` envolve o trecho de código em que uma exceção pode ser potencialmente lançada
- O lançamento de uma exceção indicando um erro é feito por meio da instrução `throw`, que instancia um objeto da classe de exceção
- O bloco `catch` é responsável pelo tratamento da exceção em questão, contendo as instruções a serem executadas quando uma exceção for lançada
  - É comum dizer que uma referência ao objeto da classe de exceção, lançado por meio do `throw` dentro de um bloco `try`, é capturado e tratado pelo bloco `catch`
  - É possível ter mais de um bloco `catch` associado a um mesmo bloco `try`, de modo que o respectivo bloco `catch` a ser executado será o correspondente ao tipo de exceção que foi lançada
  - Se uma exceção não é lançada, as instruções contidas no bloco `catch` não são executadas

Detectando e tratando exceções

Um exemplo

```
#include <iostream>
using std::cerr;
using std::cout;
using std::endl;

int main() {
    int numerador, denominador;
    cout << "Digite numerador e denominador: ";
    cin >> numerador >> denominador;

    try {
        if (denominador == 0) throw DivisaoPorZero();
        else cout << "Resultado da divisao: " << (double) numerador / denominador << endl;
    } catch (DivisaoPorZero &ex) {
        cerr << ex.what() << endl;
    } catch (...) {
        cerr << "Erro desconhecido na divisao." << endl;
    }
    return 0;
}
```

Exceção (objeto da classe `DivisaoPorZero`) é lançada se o denominador for igual a zero

O método `what` é invocado para exibir a mensagem de erro para o usuário

Exceções em alocação dinâmica de memória

Um tipo particular de exceção é o lançado quando o programa **falha** em alocar memória dinamicamente

- A exceção lançada é um objeto da classe `std::bad_alloc`, definida na biblioteca `<new>`
- Uma falha na alocação dinâmica de memória com o operador `new` lança de forma **implícita** uma exceção `bad_alloc`, que pode ser tratada por um bloco `catch`

Exceções em alocação dinâmica de memória

Um exemplo

```
#include <iostream>
using std::cerr;
using std::endl;

#include <new>
using std::bad_alloc;

int main() {
    int** p = new int*[50];
    try {
        for (int i = 0; i < 50; i++) {
            p[i] = new int[2000000000];
        }
    } catch (bad_alloc &ex) {
        cerr << ex.what() << " Erro em alocacao dinamica de memoria" << endl;
        return 1;
    }
    return 0;
}
```

Este trecho pode lançar uma exceção `bad_alloc` caso não seja possível realizar essa alocação dinâmica de memória

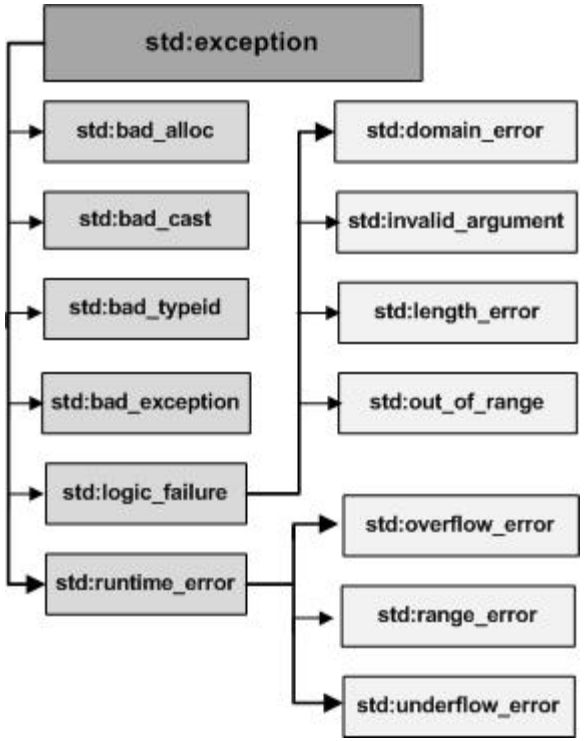
Lançada a exceção, o programa exibe a mensagem de erro e é encerrado

Exceções padrão

- Existe na STL uma série de exceções padrão, bastando incluir o cabeçalho `<stdexcept>` no código para utiliza-las
  - Recomenda-se que as exceções derivem de `std::logic_failure` ou uma de suas filhas, de acordo com a árvore de exceções padrão
- Todas as exceções padrão possuem uma descrição
- O método `what` é o responsável por retornar a mensagem de erro
- Uma das vantagens de usar uma das exceções padrão como base é que há boas chances de que o programador que usar seu código tenha fornecido algum bloco `try/catch` para tratá-las
  - Isso é especialmente valioso no caso de desenvolvimento de uma ***lib*** ou ***engine*** própria

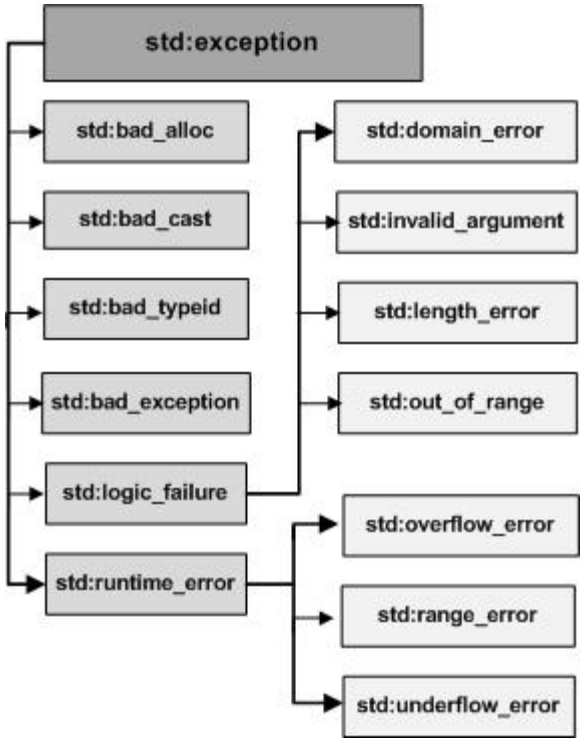
Exceções padrão

- `exception`
  - É a classe mãe de todas as exceções.
  - Uma alternativa melhor que o `catch (...)` pois ainda será possível obter o erro da `exception`.
- `bad_alloc`
  - Disparada automaticamente pelo comando `new` quando não há memória suficiente.
- `bad_cast`
  - Disparada pelo `dynamic_cast` caso não seja possível realizar a conversão de tipos em questão
- `runtime_error`
  - Exceções que denotam erros de programação comuns, que não poderiam ser previstos inicialmente pelo programador (por exemplo, no meio da avaliação de uma função, um cálculo ter o limite superior de uma variável estourada, `std::overflow_error`).

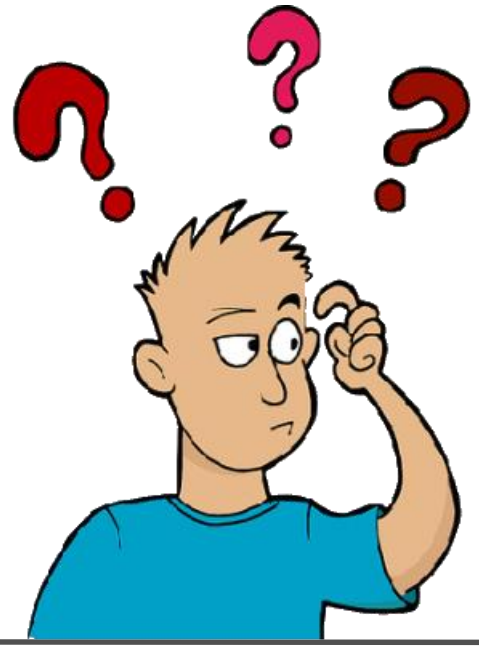


Exceções padrão

- `logic_failure`
  - São exceções que violam condições no programa ou denotam erros de programação (por exemplo, numa função de fatorial, passar valores negativos como parâmetro)
- `domain_error`
  - Indica que um erro de domínio foi cometido, ou seja, um valor incorreto foi passado para uma função matemática (por exemplo, na função de raiz quadrada, foi passado um número negativo).
- `invalid_argument`
  - Indica que o valor de um dos parâmetros recebidos pela função é inválido (por exemplo, o usuário passou um ponteiro nulo onde deveria passar um objeto)
- `length_error`
  - Indica que não é possível alterar o tamanho de uma estrutura para o valor indicado (por exemplo, o usuário tentou dar um *resize* para um tamanho maior que o máximo ou negativo)
- `out_of_range`
  - Indica que um parâmetro foi passado fora do intervalo permitido.



Alguma Questão?



Exercício

O problema da **busca em um vetor sequencial** pode ser definido como:

> *Dado um conjunto de valores previamente armazenados em um vetor V nas posições* $V[l]$, $V[l + 1]$, ..., $V[r]$, *sendo* $0 \leq l \leq r \in \mathbb{N}^0$, *verificar se um valor (chave)* k *está entre esse conjunto de valores. Em caso positivo, indicar qual o índice da localização de* k *em* V *ou retornar -1 caso* k *não seja encontrado.*

Modifique uma implementação sua de um algoritmo de busca a fim de que sejam lançadas e tratadas exceções ao invés de retornar o valor -1 em caso de falha. Teste sua implementação para casos de sucesso e falha na busca.