Exceções

BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP
Baseado nos slides do Prof. Marco Antônio Carvalho

Introdução

- Uma **exceção** é uma indicação de um **problema que ocorre** durante a **execução** de um programa
- O próprio nome indica que o problema é infrequente;
- A regra é que o programa execute corretamente.

Introdução (cont.)

- O **tratamento de exceções permite** que os **programas** sejam **mais robustos** e também **tolerantes a falhas**
- Os **erros** de execução **são processados**;
- O programa **trata o problema** e **continua executando** como se nada tivesse acontecido;
- Ou pelo menos, termina sua execução elegantemente.

Introdução (cont.)

- Para estarmos aptos a **construir um sistema robusto**, os **métodos** devem **sinalizar** todas as **condições anormais**.
- Os **métodos** devem **gerar exceções** que possam ser tratadas **para resolver ou contornar as falhas**.

Introdução (cont.)

- Considere o pseudocódigo:

```
Realize uma tarefa
Se a tarefa precedente não executou corretamente
    Realize processamento de erro
Realize a próxima tarefa
Se a tarefa precedente não executou corretamente
    Realize processamento de erro
```

Introdução (cont.)

- Considere o pseudocódigo:

```
Realize uma tarefa
Se a tarefa precedente não executou corretamente
    Realize processamento de erro
Realize a próxima tarefa
Se a tarefa precedente não executou corretamente
    Realize processamento de erro
```

Alerta

Mistura de lógica e tratamento de erro pode tornar o programa difícil de ler/depurar

Introdução (cont.)

- Tratamento de exceção **remove correção de erro da "linha principal"** do programa
  - Torna o **programa mais claro** e melhora a manutenção
  - Programadores podem decidir se **tratam todas** as excepções **ou algumas** de um tipo específico
  - Objetos de classes específicas tratam os erros: possibilidade do uso de **herança e polimorfismo**

Introdução (cont.)

- Só **pode tratar erros síncronos**:
  - Aqueles que seguem a "linha de execução" do programa
  - Exs.: divisão por zero, ponteiro nulo

Introdução (cont.)

- **Não pode** tratar **erros assíncronos** (independente do programa)
  - Ex.: I/O de disco, mouse, teclado, mensagens de rede que ocorrem em paralelo e de maneira independente do fluxo de controle do programa em execução
- Erros mais fáceis de tratar

Tratamento de Exceção

- Terminologia
  - Função que tem erros dispara uma exceção (*throws an exception*)
  - Tratamento de exceção (se existir) pode lidar com problema
    - Pega (*catches*) e trata (*handles*) a exceção
  - Se não houver tratamento de exceção, exceção não é pega
    - Pode terminar o programa (*uncaught*)

Tratamento de Exceção (cont.)

- Envolve três conceitos:
  - Utilização de um bloco tentar (*try*)
  - Captura da uma exceção por um manipulador de exceções (*catch*)
  - Disparo de uma exceção (*throw*)

Tratamento de Exceção (cont.)

- Envolver o bloco suspeito de gerar exceção com um *try*.
  - Começa com um comando *try* seguido do código entre chaves
  - Pode ser seguido de um ou mais *catch*

Tratamento de Exceção (cont.)

- A **captura** de uma exceção é feita por um *catch*.
  - Começa com a palavra *catch* seguido por uma declaração de tipo associada ao tipo de exceção a qual responde
  - Corpo igual a uma função ou método normal
  - Será disparado por um *throw*

Tratamento de Exceção (cont.)

- *throw* indica o disparo de uma exceção
  - Surge algum problema implicando o desvio para um outro ponto (bloco *catch*)
  - É seguido de um valor (string, constante, obj, etc) que indica o tipo de exceção

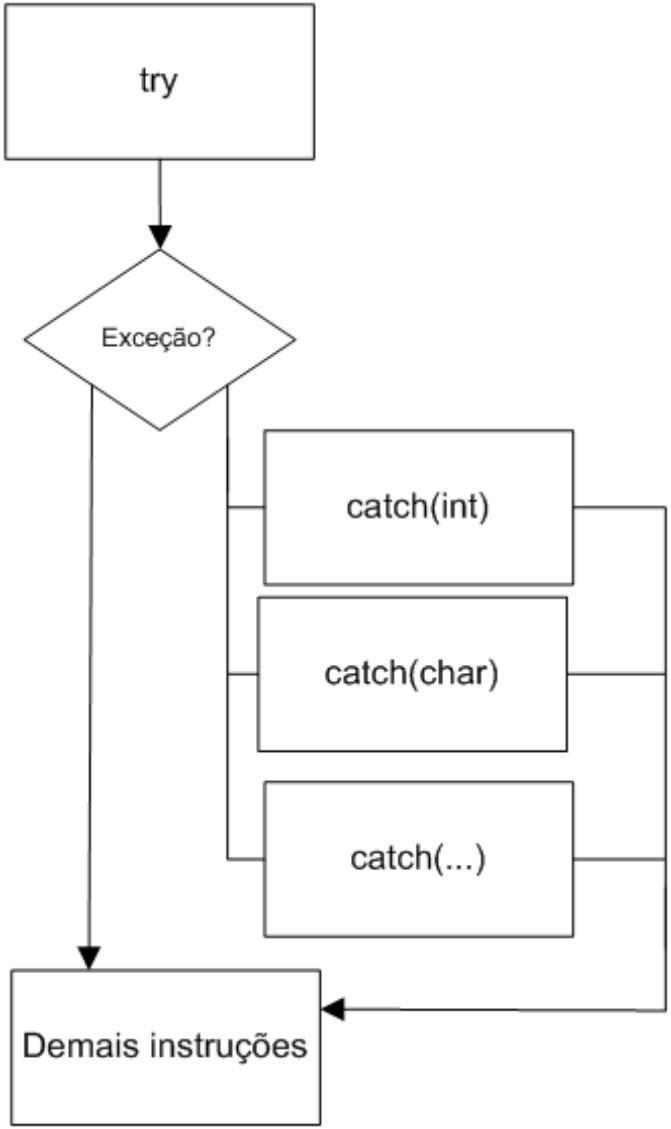
Tratamento de Exceção (cont.)

- Código C++

```
try {
    código que pode provocar uma exceção
}
catch (exceptionType){
    código para tratar a exceção
}
```

- Bloco *try* possui código que pode provocar exceção
- **Um ou mais blocos** *catch* devem ser escritos imediatamente após o bloco *try* correspondente

Tratamento de Exceção (cont.)



Bloco *try*

- utilizada para definir blocos de código em que exceções possam ocorrer
- Precedido pela palavra *try* e é delimitado por { e };
  - As instruções que podem implicar em exceções e
  - todas as instruções que não podem ser executadas em caso de exceção fazem parte do bloco de código.

Bloco *catch*

- Exceção é tratada em um bloco *catch* apropriado
  - Blocos *catch* **definem** exatamente o **tipo de exceção tratada**
  - Pode ser o **tipo exato ou uma classe base** da exceção disparada
- Parâmetro de recebimento do bloco *catch*
  - Se nomeado, pode acessar objeto de exceção

Bloco *catch* (cont.)

- Cada bloco *catch* trata apenas um tipo de exceção
  - Colocar mais de um tipo separado por vírgulas é erro de sintaxe
- Reporta a exceção ao usuário
- Termina o programa corretamente
  - Ou tenta uma estratégia alternativa para lidar com a tarefa que falhou

Bloco *throw*

- A instrução *throw* lança uma exceção
  - Indica que houve um erro;
  - É criado um objeto, que contém a informação sobre o erro
  - Logo, podemos criar uma classe que defina o erro ou utilizar uma existente.

Bloco *throw* (cont.)

- Posteriormente, outro trecho de código capturará este objeto e tomará a atitude adequada.
  - As instruções que podem implicar em exceções e
  - todas as instruções que não podem ser executadas em caso de exceção fazem parte do bloco de código.

Tratamento de Exceção

- *Throw point*
  - Local no bloco *try* onde a exceção ocorre
- Se a exceção for tratada
  - Programa pula o restante do bloco *try*
  - Executa o bloco *catch* correspondente
  - Reinicia depois do bloco *catch*

Tratamento de Exceção (cont.)

- Se a exceção for disparada mas não for tratada por nenhum bloco *catch*
- Ou se a exceção for disparada em uma sentença que não está em um bloco
  - Função termina imediatamente e
  - o programa tenta encontrar o bloco *try* na função chamadora

Tratamento de Exceção (cont.)

- Se não houver exceção
  - Programa termina o bloco *try* e continua a execução após pular todos os blocos catchs
  - Não implica queda de desempenho

Tratamento de Exceção (cont.)

Realizar a divisão de dois números digitados.

Tratamento de Exceção (cont.)

```
int main()
{
    int a, b;
    double c;

    cin >> a >> b;

    try{
        if (b == 0)
            throw "Divisao por zero \n";
        c = static_cast<double>(a) / b;
        cout << "Resposta: " << c << endl;
    }
    catch (const char* e){
        cerr << "Erro: " << e;
    }
    return 0;
}
```

Tratamento de Exceção (cont.)

Criar uma função que encontre o fatorial de um número. Gerar uma exceção quando o numero fornecido como parâmetro, é negativo

Tratamento de Exceção (cont.)

```
int fatorial(int n)
{
    if (n < 0)
        throw "Numero negativo! \n";
    int f = 1;
    for (int i = 1; i <= n; i++)
        f *= i;
    return f;
}
```

Tratamento de Exceção (cont.)

```
int main()
{
    try{
        cout << "5! = " << fatorial(5) << endl;
        cout << "-5! = " << fatorial(-5) << endl;
     }
     catch (const char* e){
        cerr << "Erro: " << e;
     }
     return 0;
}
```

Exceção usando classes

Criar a classe que trate a exceção da divisão por zero, identificando a linha onde acontece o erro.

Exceção usando classes (cont.)

```
class DivisionByZero{
    string msg;
    int line;
public:
    DivisionByZero(const string& msg, const int line) :
        msg(msg), line(line) {}
    string what() const {
        return msg + " na linha " + to_string(line);
    }
};
```

Exceção usando classes (cont.)

```
int main()
{
    int a, b;
    try {
        cin >> a >> b;
        if (b == 0)
            throw DivisionByZero("Divisao por zero",
                __LINE__);
        cout << static_cast<double>(a) / b;
    }
    catch (DivisionByZero& e){
        cerr << "Erro: " << e.what();
    }
    return 0;
}
```

FIM