



PCS311

Laboratório de Programação Orientada a Objetos para Engenharia Elétrica

Aula 9: Programação Defensiva e Exceções em C++

Escola Politécnica da Universidade de São Paulo

Agenda

1. Programação defensiva
2. Exceções em C++

Compilação

- O compilador é um programa que transforma um código de uma linguagem para outra
- O compilador faz algumas verificações
 - *Exemplo:*
 - Palavras válidas
 - Ordem das palavras está correta
 - A variável foi declarada anteriormente
 - ***Nem todos os problemas são encontrados durante a compilação***

Exemplo

- Quais problemas o seguinte código pode ter?

```
...
11 void Disciplina::adicionarAluno (string nome, double notaP1,
12                                   double notaP2,
13                                   double notaP3, int faltas) {
14     this->alunos[numeroDeAlunos++] = new Aluno (nome, notaP1,
15                                                  notaP2, notaP3, faltas);
16     numeroDeAlunos++;
17 }
```

EX01

```
...
8  Aluno::Aluno (string nome, double notaP1, double notaP2,
9               double notaP3, int faltas) {
10     this->nome = nome;
11     this->notaP1 = notaP1;
12     this->notaP2 = notaP2;
13     this->notaP3 = notaP3;
14     this->faltas = faltas;
15 }
```

EX01

Exemplo

- Alguns impactos (nesse programa)
 - Uso de posições inválidas do vetor
 - Alteração de outras variáveis (talvez até em outras classes)
 - Perda da informação armazenada
 - Cálculos incorretos
 - Média incorreta (um outro método)
 - Programa termina / trava inesperadamente
 - Apresentação de informação errada
- Como lidar com isso?
 - **Programação defensiva**

Programação defensiva

- Código se proteger de **entradas** inadequadas
 - Mesmo que não seja *culpa do seu código*
 - Similar à *direção defensiva*
 - *...programas terão problemas e modificações...*
- Cuida de eventos que *não deveriam* acontecer
 - Ou pior: ***nunca*** deveriam acontecer

Programação defensiva

■ *Preocupações*

- Verificar os valores dos parâmetros de entrada
- Verificar dados obtidos de fontes *externas*
 - (Fora da classe)

Parâmetros *inadequados*

```
#include "C.h"
...
void C::x(A *a, B *b) {
    ...
    C *c = ???;
    ...
}
```

- Classes
- Usuário
- Arquivos
- Outros sistemas

- Decidir *como lidar* com entradas com problemas

Técnicas de tratamento de erro


- Algumas opções para lidar com erros
 - **Retornar um código de erro**
 - Terminar o programa
 - Registrar em um log o problema
 - Arquivo de log
 - Retornar um valor neutro
 - *Exemplo:* método que desenha algo em tela
 - Retornar a mesma resposta que da última vez
 - *Exemplo:* um sensor de temperatura
 - Apresentar uma mensagem de erro ao usuário

A decisão de qual opção considerar depende do problema em questão e do sistema

Retornar um código de erro

- Reporta o erro e permite que quem chamou o método decida o que fazer
 - *Exemplo:* método adicionarAluno

Verdadeiro se foi possível adicionar ou falso caso contrário



```
1 bool Disciplina::adicionarAluno (string nome, double notaP1,  
2                                 double notaP2, double notaP3,  
3                                 int faltas) {  
4     if (numeroDeAlunos >= NUMERO_MAXIMO) return false;  
5  
6     this->alunos[numeroDeAlunos++] = new Aluno (nome, notaP1,  
7         notaP2, notaP3, faltas);  
8     numeroDeAlunos++;  
9     return true;  
10 }
```

Retornar um código de erro

- O que fazer se o método já tiver um retorno e/ou não posso alterá-lo?
 - *Exemplo*

```
8  Aluno::Aluno (string nome, double notaP1, double notaP2,  
9               double notaP3, int faltas) {  
10     this->nome = nome;  
11     this->notaP1 = notaP1;  
12     this->notaP2 = notaP2;  
13     this->notaP3 = notaP3;  
14     this->faltas = faltas;  
15 }
```

EX01

- O que fazer se o método não souber o que retornar (ou o que fazer)?
- **Solução:** uso de exceções

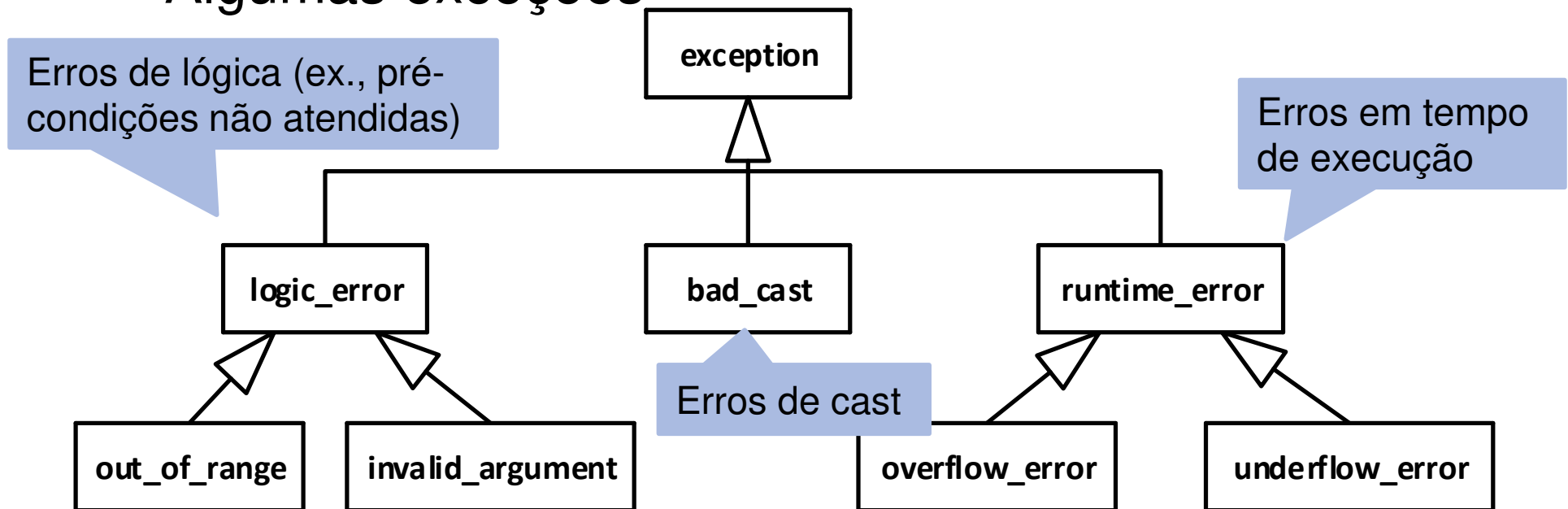
Exceções em C++

Exceção

- Um evento que causa a suspensão da execução *normal* de um programa
 - Em geral, algo que *não deveria acontecer*
 - Situações **excepcionais**
- Métodos podem **jogar** (**throw**) exceções
- Exceções são **capturadas** (**catch**) e tratadas
 - O método que chamou (direta ou indiretamente) pode capturar a exceção
 - Outro método ou o programa principal pode capturá-la
 - Permite continuar a execução do programa
- Não é algo específico da Orientação a Objetos

Exceção em C++

- A biblioteca padrão define exceções básicas
 - Necessário `#include <stdexcept>`
 - É necessário `using namespace std;`
 - Algumas exceções



- O método `what()` de `exception` possui o motivo da exceção

Jogando uma exceção em C++

■ Comando throw

- Pode jogar **objetos** ou **tipos primitivos**
- Sintaxe `throw <objeto>;`

■ *Exemplo*

```
...
8  Aluno::Aluno (string nome, double notaP1, double notaP2,
9                double notaP3, int faltas) {
10     if (nome.empty()) throw new invalid_argument ("Nome vazio");
11     else if (notaP1 < 0 || notaP1 > 10 || notaP2 < 0
12             || notaP2 > 10 || notaP3 < 0 || notaP3 > 10)
13         throw new invalid_argument ("Nota invalida");
14     else if (faltas < 0)
15         throw new invalid_argument ("Falta negativa");
16
17     this->nome = nome;
18     this->notaP1 = notaP1;
19
20     ...
22 }
```

EX02

Capturando uma exceção em C++

■ Bloco try-catch

- Tenta executar o código e captura uma exceção se ela acontecer

```
try {  
    //...  
    //...  
} catch (<Exceção A> e) {  
    //...  
    //...  
} catch (<Exceção B> e) {  
    //...  
}
```

The diagram shows four arrows pointing from the code to their explanations:

- From `//...` to "Tenta executar o código"
- From `catch (<Exceção A> e)` to "Captura a exceção que pode acontecer"
- From `//...` to "Código caso a exceção ocorra"
- From `catch (<Exceção B> e)` to "Captura outro tipo de exceção"

- Pode-se ter vários "catch"
 - Um para cada tipo de exceção

Capturando uma exceção em C++

■ Exemplo

```
...
9      try {
10         Disciplina *d = new Disciplina();
11         d->adicionarAluno ("Ana", 10, 10, 10, -1);
12         d->adicionarAluno ("Joao", 5, 5, 5, 3);
13
14         d->imprimir();
15     } catch (invalid_argument *e) {
16         cout << "Erro: " << e->what();
17         delete e; // limpando!
18     }
```

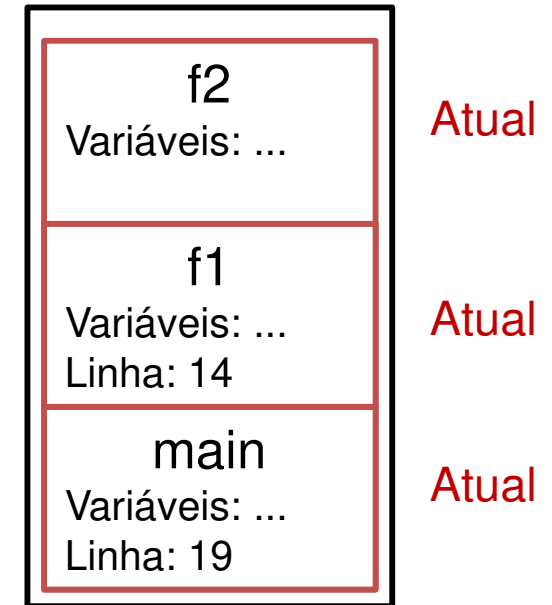
EX02

Capturando uma exceção em C++

- Quem deve capturar a exceção?
 - Depende: **deve ser quem sabe como tratá-la!**
 - (Nem sempre é quem chamou o método diretamente)
- O que acontece se uma exceção não for capturada?
 - Quando um método não captura a exceção, o **contexto** dele na pilha de execução é retirado
 - Ela é jogada para o **próximo contexto** da pilha de execução
 - Se ninguém capturar (nem o main), o **programa termina**

Pilha de execução

EX03



Pilha de execução

```
1  #include <iostream>
2  #include <stdexcept>
3
4  using namespace std;
5
6  int f2 (int a) {
7      if (a == 0)
8          throw new invalid_argument ("\"a\" nao e valido");
9
10     return a + 1;
11 }
12
13 int f1 (int a) {
14     return f2 (a) + 1;
15 }
16
17 int main (int argc, char **argv) {
18     try {
19         cout << "Resultado: " << f1 (0) << endl;
20     } catch (invalid_argument *e) {
21         cout << "Erro: " << e->what() << endl;
22         delete e;
23     }
24     return 0;
25 }
```

Boas práticas

- Não jogue uma exceção se o próprio método pode tratá-la
- Saiba as exceções que a classe usada joga
- Use a exceção mais adequada
 - Tente usar as exceções da biblioteca padrão
 - Não use a exceção `pai exception`
 - Ela nem tem um construtor que recebe uma `string`!
 - **Crie uma nova exceção** se não houver uma adequada
 - O ideal é que ela seja filha de algum tipo de `exception`

Criando uma classe para a exceção

```
...
3  #include <stdexcept>
4  #include <string>
5
6  using namespace std;
7
8  class ErroDisciplina : public runtime_error {
9  public:
10     ErroDisciplina (string mensagem);
11 };
```

EX04

```
1  #include "ErroDisciplina.h"
2
3  ErroDisciplina::ErroDisciplina (string mensagem) :
4      runtime_error (mensagem) {}
```

EX04

Criando uma classe para a exceção

```
...  
47 double Disciplina::media (int numeroDeFaltasMaximo) {  
48     if (numeroDeFaltasMaximo < 0)  
49         throw new ErroDisciplina ("Numero de faltas e' < 0");  
50     else if (numeroDeAlunos == 0)  
51         throw new ErroDisciplina ("Nao existem alunos matriculados");  
...  
65     return total / alunosPresentes;  
66 }
```

Joga a exceção
(cria um objeto e joga)

EX04

■ Outras exceções e detalhes

- <http://www.cplusplus.com/reference/exception/exception/>

Boas práticas

- Cuidado ao jogar exceções em construtores
 - Alguma inicialização pode não ter sido feita
 - *O destrutor não será chamado automaticamente*
 - *Mas se algo foi alocado, ele precisaria ser desalocado!*
 - (Destrutores não jogam exceções)
- Destrua o objeto de exceção se criado com **new**
- Não crie blocos catch vazios
 - Eles “engolem” exceções

Conclusão

- Programação defensiva tem um custo
 - Perda de desempenho por causa do excesso de verificações
- Outras verificações podem ser úteis durante o desenvolvimento
 - *Asserções*
 - Permitem verificar o programa durante a sua execução
 - Condições que **nunca** *deveriam* acontecer
 - **Terminam o programa** quando falham
 - Devem ser desabilitadas *em operação (produção)*

Conclusão

- Em um código em *produção* (operação)
 - Deixe verificações para erros importantes
 - Remova verificações para erros triviais
 - Remova código que causa o término do programa forçosamente
 - Faça um *registro* de erros para facilitar a correção
 - Log: depuração
 - Verifique se as mensagens de erro são adequadas para o usuário
- **Prefira exceções a códigos de erro**
 - Uma exceção deixa mais claro o problema

Bibliografia

- MCCONNELL, S. **Code Complete**. 2nd edition. Microsoft Press, 2004. Capítulo 8.