

# Linguagens de Programação

Prof. Miguel Elias Mitre Campista

<http://www.gta.ufrj.br/~miguel>

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

# Parte II

Introdução à Programação em C++  
(Continuação)

## Relembrando da Última Aula...

- Entrada e saída
- Mais exemplos de programação orientada a objetos...

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tratamento de Exceção

- Exceções
  - Indicam problemas ocorridos no programa
    - Ocorrências nem sempre esperadas que não deveriam acontecer
  - Representam comportamento que não é comum
    - Uma "exceção" em um programa que normalmente funciona

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tratamento de Exceção

- Tratamento de exceção
  - Programas que resolvem exceções
    - Continuam a sua execução mesmo em face de um erro
  - Programas que são capazes de continuar execução
    - Término controlado
      - Problemas mais severos podem impedir que um programa continue a sua execução
  - Programas que toleram falhas
    - Ex.: Lidar com um programa que divida por zero

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tratamento de Exceção

- 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*

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tratamento de Exceção

- 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*

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

## Tratamento de Exceção

- 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 exceções
    - Exceções de um tipo específico
    - Exceções de tipos relacionados
  - Objetos de classes específicas tratam os erros
    - Possibilidade do uso de **herança e polimorfismo**

## Tratamento de Exceção

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

- 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

## 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
- Cada bloco `catch` trata apenas um tipo de exceção
  - Colocar mais de um tipo separado por vírgulas é erro de sintaxe

## Bloco catch

- Entre outras funções, ele pode:
  - Reportar a exceção ao usuário
  - Registrar a exceção em um arquivo
  - Terminar o programa corretamente
    - Ou tentar uma estratégia alternativa para lidar com a tarefa que falhou

## 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`
      - Variáveis locais ao bloco `catch` saem do escopo

Execução do programa não retorna ao ponto onde a exceção foi disparada!

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tratamento de Exceção

- **Throw point**
  - 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
- Se não houver exceção
  - Programa termina o bloco `try` e continua a execução após pular todos os blocos `catch`'s
    - Não implica queda de desempenho

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Outras Técnicas para Tratamento de Erros

- Ignorar exceção
  - Típico para software pessoal (não comercial)
  - Programa pode falhar
- Abortar programa
  - Frequentemente apropriado
  - Não é apropriado para software de missão crítica
- Teste para condição de erro
  - Chamar função `exit (<cstdlib>)` e passar código de erro

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo Simples de Tratamento de Exceção: Divisão por Zero

- Palavra-chave: **throw**
  - Dispara uma exceção
    - Usada quando ocorre erro
  - Pode disparar objeto de exceção, inteiro etc.
    - `throw myObject;`
    - `throw 5;`
- Objetos de exceção
  - Classe base exceção (`<exception>`)
  - Construtor pode receber uma string (para descrever a exceção)
    - Função membro `what ()` retorna essa string

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <string>
#include <exception>

using namespace std;

int main() {
    string n = "excecao";
    try {
        throw n;
    }
    catch (string e) {
        cout << e << endl;
    }
    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <string>
#include <exception>

using namespace std;

int main() {
    string n = "excecao";

    // ...

    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo Simples de Tratamento de Exceção: Divisão por Zero

- Tratamento de erros por divisão por zero
  - Define nova classe de exceção
    - `DivideByZeroException`
    - Herdada da classe `exception`
  - Na função de divisão
    - Testar denominador
    - Se zero, dispara uma exceção (throw object)
  - No bloco try
    - Tentativa de dividir
    - Possui associado o bloco catch
      - Pega objetos `DivideByZeroException`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Primeiro Exemplo Usando Tratamento de Exceção em C++

**Possibilidade 1**

```
/*
 * Aula 16 - Exemplo 1
 * Programa erroCap16Ex1.h
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>

using namespace std;
//using std::exception;

// Objetos DivideByZeroException devem ser disparados por funções
// assim que detectada a exceção de divisão por zero
class DivideByZeroException : public exception {
public:
    // construtor especifica mensagem padrão de erro
    DivideByZeroException():DivideByZeroException()
    : exception() {}
    virtual const char* what() const throw() {
        return "attempted to divide by zero";
    }
};
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Primeiro Exemplo Usando Tratamento de Exceção em C++

**Possibilidade 2**

```
/*
 * Aula 16 - Exemplo 1
 * Programa erroCap16Ex1.h
 * Autor: Miguel Campista
 */
#include <iostream>
#include <stdexcept>

using namespace std;
//using std::exception;

// Objetos DivideByZeroException devem ser disparados por funções
// assim que detectada a exceção de divisão por zero
class DivideByZeroException : public runtime_error {
public:
    // construtor especifica mensagem padrão de erro
    DivideByZeroException():DivideByZeroException()
    : runtime_error("attempted to divide by zero") {}
};
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Primeiro Exemplo Usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 1
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "erroCap16Ex1.h"

// realiza divisão e dispara objeto DivideByZeroException object se
// uma exceção de divisão por zero ocorrer
double quotient(int numerator, int denominator) {
    // dispara DivideByZeroException se tentar dividir por zero
    if (denominator == 0)
        throw DivideByZeroException(); // termina a função
    // retorna resultado da divisão
    return static_cast<double>( numerator ) / denominator;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Primeiro Exemplo Usando Tratamento de Exceção em C++

```
int main() {
    int number1; // numerador definido pelo usuário
    int number2; // denominador definido pelo usuário
    double result; // resultado da divisão

    cout << "Enter two integers (end-of-file to end): ";

    // solicita ao usuário entrar com dois números
    while (cin >> number1 >> number2) {

        // bloco try contém código que pode disparar exceção
        // o código que deve não executar se a exceção ocorrer
        try {
            result = quotient( number1, number2 );
            cout << "The quotient is: " << result << endl;
        }

        // tratador de exceção trata uma exceção de divisão por zero
        catch ( DivideByZeroException &divideByZeroException ) {
            cout << "Exception occurred: "
                << divideByZeroException.what() << endl;
        }

        cout << "\nEnter two integers (end-of-file to end): ";
    }
    cout << endl;

    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Primeiro Exemplo Usando Tratamento de Exceção em C++

```
int main() {
    int number1; // numerador definido pelo usuário
    int number2; // denominador definido pelo usuário
    double result; // resultado da divisão
    cout << "Entre two integers (end-of-file to end): ";

    // solicita ao usuário entrar com dois números
    // C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projeto\aula16-ex1.exe
    Enter two integers (end-of-file to end): 1 2
    The quotient is: 0.5
    Enter two integers (end-of-file to end): 4 5
    The quotient is: 0.8
    Enter two integers (end-of-file to end): 5 0
    exception occurred: attempted to divide by zero
    Enter two integers (end-of-file to end): _

    cout << "\nEntre two integers (end-of-file to end): ";
    cout << endl;
    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Redisparo de uma Exceção (Rethrow exception)

- Usado quando um tratador de exceção não pode processar a exceção ou quando pode somente processá-la parcialmente

- Nesses casos, o tratador da exceção pode adiar o tratamento

- Pode re disparar mesmo após o tratador ter feito algum processamento
- Pode re disparar uma exceção para um outro tratador
  - Vai para o próximo bloco try
  - Blocos catch correspondentes tentam tratar

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Redisparo de uma Exceção (Rethrow exception)

- Para re disparar
  - Usado com o sentença "throw;"
    - Sem argumentos
    - Termina uma função

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 2
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>

using namespace std;

// dispara, trata e re dispara a exceção
void throwException() {
    // dispara exceção e a pega imediatamente
    try {
        cout << "Function throwException throws an exception\n";
        throw exception(); // gera uma exceção
    }

    // trata exceção
    catch ( exception caughtException ) {
        cout << " Exception handled in function throwException"
            << "\n Function throwException rethrows exception";

        throw; // re dispara a exceção para processamento posterior
    }
    cout << "This also should not print!\n";
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
int main() {
    // dispara exceção
    try {
        cout << "\nmain invokes function throwException\n";
        throwException();
        cout << "This should not print!\n";
    }

    // trata exceção
    catch ( exception caughtException ) {
        cout << "\nException handled in main!\n";
    }

    cout << "Program control continues after catch in main!\n";
    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
int main() {
    // dispara exceção
    try {
        cout << "\nmain invokes function throwException\n";
    }

    // C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projeto\aula16-ex3.exe
    main invokes function throwException
    Function throwException throws an exception
    Exception handled in function throwException
    Function throwException rethrows exception
    Exception handled in main
    Program control continues after catch in main
    Pressione qualquer tecla para continuar. . .
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Segundo Exemplo Usando Tratamento de Exceção em C++

- O **redisparo** fez com que...
  - A função `throwException` não continue a sua execução após o `catch`
    - Se não houvesse **redisparo a execução da função** continuaria
  - O bloco `try` da função principal não continue a sua execução após a chamada da função `throwException`
    - Se não houvesse **redisparo a execução da função** continuaria
  - O `catch` da função principal fosse invocado
    - Se não houvesse **redisparo a execução da função** continuaria

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Especificação de Exceções

- Tipo de exceção disparada por uma função pode ser limitada
  - Adição de sufixo no protótipo da função
- Se `myfunction` disparar outro tipo de exceção, essa não é tratada pelo `catch` de inteiro correspondente
  - Compilador permite disparos de tipos diferentes do definido, entretanto erros podem ocorrer em execução

```
float myfunction (char) throw (int);
```

```
// Exceções não são permitidas
float myfunction (char) throw ();
// Todas as exceções são permitidas
float myfunction (char);
```

## Especificação de Exceções

- Lista de exceções que podem ser disparadas
  - Também chamada de "lista de disparo" (*throw list*)

```
int someFunction( double value )
    throw ( ExceptionA, ExceptionB, ExceptionC ) {
    // corpo da função
}
```
  - Pode somente disparar `ExceptionA`, `ExceptionB` e `ExceptionC` (e classes derivadas)
    - Se dispara outro tipo, função `unexpected` é chamada
      - Por padrão, essa função termina o programa

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Processamento de Exceções Unexpected

- Função `unexpected`
  - Chamada quando a exceção disparada não se encontra na `throw list`
  - Chama função registrada com `set_unexpected`
    - Definida em `<exception>`
    - Caso nenhuma função tenha sido registrada, função `terminate` é chamada por padrão
  - `set_terminate`
    - Define qual função `terminate` é chamada
    - Por padrão, chama `abort`
      - Se redefinido, ainda chama `abort` depois da nova função `terminar`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Processamento de Exceções Unexpected

- Argumentos para as funções de definição: `set_unexpected` e `set_terminate`
  - Recebe ponteiro para função
    - Função não deve receber argumentos
  - Retorna `void`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 2.1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>

using namespace std;

void myunexpected () {
    cerr << "unexpected called!\n";
    throw 0; // dispara int (na especificação de exceção)
}

void myfunction () throw (int) {
    throw 'a'; // dispara char (não na especificação de exceção)
}

int main () {
    set_unexpected (myunexpected);
    try {
        myfunction();
    }
    catch (int) { cerr << "caught int!\n"; }
    return 0;
}
```

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
/*  
 * Aula 16 - Exemplo 2.1  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
  
#include <iostream>  
#include <exception>  
  
using namespace std;
```

```
int main () {  
    set_unexpected (myunexpected);  
    try {  
        myfunction();  
    }  
    catch (int) { cerr << "caught int!\n"; }  
    return 0;  
}
```

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
/*  
 * Aula 16 - Exemplo 2.1  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
  
#include <iostream>  
#include <exception>  
  
using namespace std;
```

```
void myunexpected () {  
    cerr << "unexpected called!\n";  
    throw 0; // dispara int (na especificação de exceção)  
}  
  
void myfunction () throw (int) {  
    throw 'x'; // dispara char (não na especificação de exceção)  
}
```

E se inserirmos o tipo char na throw list?

```
int main () {  
    myfunction();  
    catch (int) { cerr << "caught int!\n"; }  
    return 0;  
}
```

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
/*  
 * Aula 16 - Exemplo 2.1  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
  
#include <iostream>  
#include <exception>  
  
using namespace std;
```

```
void myunexpected () {  
    cerr << "unexpected called!\n";  
    throw 0; // dispara int (na especificação de exceção)  
}  
  
void myfunction () throw (int, char) {  
    throw 'x'; // dispara char (na especificação de exceção)  
}
```

Funciona?

```
myfunction();  
    catch (int) { cerr << "caught int!\n"; }  
    return 0;  
}
```

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
/*  
 * Aula 16 - Exemplo 2.1  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
  
#include <iostream>  
#include <exception>
```

```
void myfunction () throw (int, char) {  
    throw 'x'; // dispara char (na especificação de exceção)  
}
```

Por que não?

```
myfunction();  
    catch (int) { cerr << "caught int!\n"; }  
    return 0;  
}
```

## Segundo Exemplo Usando Tratamento de Exceção em C++

Faltava definir o bloco catch correspondente?

```
/*  
 * Aula 16 - Exemplo 2.1  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
  
#include <iostream>  
#include <exception>  
  
using namespace std;
```

```
void myunexpected () {  
    cerr << "unexpected called!\n";  
    throw 0; // dispara int (na especificação de exceção)  
}  
  
void myfunction () throw (int, char) {  
    throw 'x'; // dispara char (na especificação de exceção)  
}
```

```
int main () {  
    set_unexpected (myunexpected);  
    try {  
        myfunction();  
    }  
    catch (int) { cerr << "caught int!\n"; }  
    catch (char) { cerr << "caught char!\n"; }  
    return 0;  
}
```

## Segundo Exemplo Usando Tratamento de Exceção em C++

```
/*  
 * Aula 16 - Exemplo 2.1  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
  
#include <iostream>  
#include <exception>
```

```
void myunexpected () {  
    cerr << "unexpected called!\n";  
    throw 0; // dispara int (na especificação de exceção)  
}  
  
void myfunction () throw (int, char) {  
    throw 'x'; // dispara char (na especificação de exceção)  
}
```

```
int main () {  
    set_unexpected (myunexpected);  
    try {  
        myfunction();  
    }  
    catch (int) { cerr << "caught int!\n"; }  
    catch (char) { cerr << "caught char!\n"; }  
    return 0;  
}
```

## Segundo Exemplo Usando Tratamento de Exceção em C++

```

/* Aula 16 - Exemplo 2.1
 * Arquivo Principal
 * Autor: Miguel Campista
 */

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

void myterminate () {
    cerr << "my terminate!\n";
    exit (0);
}

void myunexpected () {
    cerr << "unexpected called!\n";
    throw "x"; // dispara erro
}

void myfunction () throw (int) {
    throw "x"; // dispara char (não na especificação de exceção)
}

int main () {
    set_terminate (myterminate);
    set_unexpected (myunexpected);
    try {
        myfunction();
    }
    catch (int) { cerr << "caught int!\n"; }
    return 0;
}

```

Provocando erro para chamada da função definida em set\_terminate. A função não causa chamada recursiva pois a função myunexpected pode disparar qualquer exceção

## Segundo Exemplo Usando Tratamento de Exceção em C++

```

/* Aula 16 - Exemplo 2.1
 * Arquivo Principal
 * Autor: Miguel Campista
 */

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

void myterminate () {
    cerr << "my terminate!\n";
    exit (0);
}

void myunexpected () {
    cerr << "unexpected called!\n";
    throw "x"; // dispara char (provoca o erro novamente)
}

void myfunction () throw (int) {
    throw "x"; // dispara char (não na especificação de exceção)
}

int main () {
    set_terminate (myterminate);
    set_unexpected (myunexpected);
    try {
        myfunction();
    }
    catch (int) { cerr << "caught int!\n"; }
    return 0;
}

```

Término controlado, caso contrário, é chamada a função abort diretamente

## Segundo Exemplo Usando Tratamento de Exceção em C++

```

/* Aula 16 - Exemplo 2.1
 * Arquivo Principal
 * Autor: Miguel Campista
 */

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

void myterminate () {
    cerr << "my terminate!\n";
    exit (0);
}

void myunexpected () {
    cerr << "unexpected called!\n";
    throw "x"; // dispara char (provoca o erro novamente)
}

void myfunction () throw (int) {
    throw "x"; // dispara char (não na especificação de exceção)
}

int main () {
    set_terminate (myterminate);
    set_unexpected (myunexpected);
    try {
        myfunction();
    }
    catch (int) { cerr << "caught int!\n"; }
    return 0;
}

```

Término controlado, caso contrário, é chamada a função abort diretamente

```

C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula16-ed\exe
unexpected called
my terminate
Pressione qualquer tecla para continuar. . .

```

## Liberação da Pilha

- Se exceção dispara mas não é pega
  - Termina função atual
    - Libera chamada da função da pilha de execução
  - Procura try/catch que pode tratar a exceção
    - Se nenhuma for encontrada, libera novamente
- Se exceção nunca for pega
  - Chama terminate

## Terceiro Exemplo usando Tratamento de Exceção em C++

```

/*
 * Aula 16 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */

#include <iostream>
#include <stdexcept>
using namespace std;

// função0 dispara erro run-time
void function0() throw ( runtime_error ) {
    throw runtime_error( "runtime_error in function0" ); // quarto
}

// função2 invoca função0
void function2() throw ( runtime_error ) {
    function0(); // terceiro
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
    function2(); // segundo
}

```

## Terceiro Exemplo usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 3

```

Blocos try/catch não são encontrados em nenhuma das funções, a exceção das funções é terminada...

```

using namespace std;

// função0 dispara erro run-time
void function0() throw ( runtime_error ) {
    throw runtime_error( "runtime_error in function0" ); // quarto
}

// função2 invoca função0
void function2() throw ( runtime_error ) {
    function0(); // terceiro
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
    function2(); // segundo
}

```

Função 3 dispara exceção que não é pega nem na própria função, nem na função 2 e nem na função 1



## Terceiro Exemplo usando Tratamento de Exceção em C++

```
// demonstra liberação de pilha
int main() {
    // invoca função
    try {
        function1(); // primeiro
    }

    // testa erro run-time
    catch ( runtime_error e ) { // quinto
        cout << "Exceção ocorreu: " << e.what() << endl;
    }

    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Terceiro Exemplo usando Tratamento de Exceção em C++

```
// demonstra liberação de pilha
int main() {
    // invoca função
    try {
        function1(); // primeiro
    }

    // testa erro run-time
    catch ( runtime_error e ) { // quinto
        cout << "Exceção ocorreu: " << e.what() << endl;
    }

    return 0;
}
```

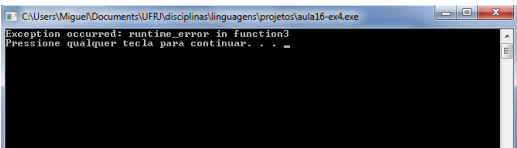
Exceção só é pega na função principal...

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Terceiro Exemplo usando Tratamento de Exceção em C++

```
// demonstra liberação de pilha
int main() {
    // invoca função
    try {
        function1(); // primeiro
    }
}
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula16-ex4.exe
Exception occurred: runtime_error in function3
Pressione qualquer tecla para continuar. . .
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Terceiro Exemplo usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <stdexcept>

using namespace std;

// função3 dispara erro run-time
void function3() throw ( runtime_error ) {
    throw runtime_error( "runtime_error in function3" ); // quinto
}

// função2 invoca função3
void function2() throw ( runtime_error ) {
    try {
        function3();
    } // terceiro
    catch ( runtime_error e ) {
        cout << "E2" << endl;
    }
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
    function2(); // segundo
    cout << "E1\n";
}
```

E se fosse assim?

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Terceiro Exemplo usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <stdexcept>

using namespace std;
```

```
// função3 dispara erro run-time
void function3() throw ( runtime_error ) {
    throw runtime_error( "runtime_error in function3" ); // quinto
}

// função2 invoca função3
void function2() throw ( runtime_error ) {
    try {
        function3();
    } // terceiro
    catch ( runtime_error e ) {
        cout << "E2" << endl;
        throw;
    }
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
    function2(); // segundo
    cout << "E1\n";
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Terceiro Exemplo usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <stdexcept>

using namespace std;
```

```
// função3 dispara erro run-time
void function3() throw ( runtime_error ) {
    throw runtime_error( "runtime_error in function3" ); // quinto
}

// função2 invoca função3
void function2() throw ( runtime_error ) {
    try {
        function3();
    } // terceiro
    catch ( runtime_error e ) {
        cout << "E2" << endl;
        throw;
    }
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
    function2(); // segundo
    cout << "E1\n";
}
```

E se fosse assim?

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Terceiro Exemplo usando Tratamento de Exceção em C++

```
..
* Aula 16 - Exemplo 3
* Arquivo Principais
* Autor: Miguel Campista
..
#include <iostream>
#include <stdexcept>
using namespace std;

// Função que gera erro run-time
void function3() throw ( runtime_error ) {
    cout << "3" << endl;
}

// Função que chama function3
void function1() throw ( runtime_error ) {
    function3(); // segundo
    cout << "result"
}


```

## Construtores, Destrutores e Tratamento de Exceção

- Erro no construtor
  - new falha
    - Por exemplo: não pode alocar memória
  - Construtor não pode retornar um valor: Como informar o usuário?
    - Espera-se que o usuário examine o objeto e note os erros?
    - Uso de variáveis globais?
  - Boa alternativa: disparar uma exceção
    - Liberação da pilha

## Quarto Exemplo usando Tratamento de Exceção em C++

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

class Cadastro {
public:
    Cadastro () throw (runtime_error) {
        throw runtime_error ("erro no construtor\n");
    }
};

int main() {
    try {
        Cadastro *cad = new Cadastro;
    }
    catch (runtime_error e) {
        cout << e.what ();
    }
    return 0;
}


```

## Quarto Exemplo usando Tratamento de Exceção em C++

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

class Cadastro {
public:
    Cadastro () throw (runtime_error) {
        throw runtime_error ("erro no construtor\n");
    }
};

int main() {
    try {
        Cadastro *cad = new Cadastro;
    }
    catch (runtime_error e) {
        cout << e.what ();
    }
    return 0;
}


```

## Exceções e Herança

- Classes de exceção
  - Podem ser derivadas de uma classe base
    - Por exemplo, **exception**
  - Se catch pode tratar classe base, pode tratar classes derivadas
    - Programação polimórfica

## Processamento de Novas Falhas

- Quando o new falha para alocar memória...
  - Deve-se disparar exceção do tipo **bad\_alloc**
    - Definida em **<new>**
  - Alguns compiladores têm new retornando 0 (zero)
  - Resultado depende do compilador

## Quarto Exemplo usando Tratamento de Exceção em C++

```

/*
 * Aula 16 - Exemplo 4
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
    double *ptr[ 100 ];

    // aloca memória para ptr
    for ( int i = 0; i < 100; i++ ) {
        ptr[ i ] = new double[ 5000000 ];

        // dev retorna 0 em caso de falha para alocar memória
        if ( ptr[ i ] == 0 ) {
            cout << "Memory allocation failed for ptr["
                << i << "]!\n";
            break;
        } else
            // alocado bem sucedido de memória
            cout << "Allocated 5000000 doubles in ptr["
                << i << "]!\n";
    }

    return 0;
}

```

## Quarto Exemplo usando Tratamento de Exceção em C++

```

/*
 * Aula 16 - Exemplo 4
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

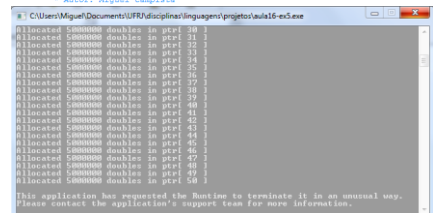
int main() {
    double *ptr[ 100 ];

    // aloca memória para ptr
    for ( int i = 0; i < 100; i++ ) {
        ptr[ i ] = new double[ 5000000 ];

        // dev retorna 0 em caso de falha para alocar memória
        if ( ptr[ i ] == 0 ) {
            cout << "Memory allocation failed for ptr["
                << i << "]!\n";
            break;
        } else
            // alocado bem sucedido de memória
            cout << "Allocated 5000000 doubles in ptr["
                << i << "]!\n";
    }

    return 0;
}

```



## Quinto Exemplo usando Tratamento de Exceção em C++

```

/*
 * Aula 16 - Exemplo 5
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <new>

using namespace std;

int main() {
    double *ptr[ 100 ];

    try {
        // tenta alocar memória
        // aloca memória para ptr; dev disparar bad_alloc em caso de falha
        for ( int i = 0; i < 100; i++ ) {
            ptr[ i ] = new double[ 5000000 ];
            cout << "Allocated 5000000 doubles in ptr["
                << i << "]!\n";
        }
    }

    // trata exceção bad_alloc
    catch ( bad_alloc &memoryAllocationException ) {
        cout << "Exception occurred: "
            << memoryAllocationException.what() << endl;
    }

    return 0;
}

```

## Quinto Exemplo usando Tratamento de Exceção em C++

```

/*
 * Aula 16 - Exemplo 5
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <new>

using namespace std;

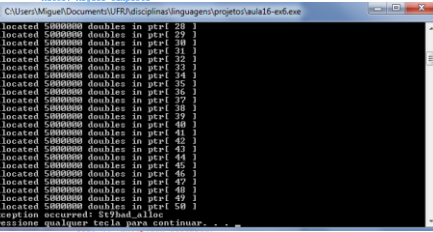
int main() {
    double *ptr[ 100 ];

    try {
        // tenta alocar memória
        // aloca memória para ptr; dev disparar bad_alloc em caso de falha
        for ( int i = 0; i < 100; i++ ) {
            ptr[ i ] = new double[ 5000000 ];
            cout << "Allocated 5000000 doubles in ptr["
                << i << "]!\n";
        }
    }

    // trata exceção bad_alloc
    catch ( bad_alloc &memoryAllocationException ) {
        cout << "Exception occurred: "
            << memoryAllocationException.what() << endl;
    }

    return 0;
}

```



## Processamento de Novas Falhas

- **set\_new\_handler**
  - Cabeçalho <new>
  - Registra função para chamar quando new falha
  - Usa ponteiro de função para funções que:
    - Não possui parâmetros
    - Retorna void
  - Uma vez registrada, função é chamada ao invés de disparar exceção

## Sexto Exemplo usando Tratamento de Exceção em C++

```

/*
 * Aula 16 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <new>
#include <cstdlib>

using namespace std;

void customNewHandler() {
    cout << "customNewHandler was called";
    abort();
}

// Usando set_new_handler para tratar falha na alocação de memória
int main() {
    double *ptr[ 100 ];

    // especifica que customNewHandler deve ser chamada em caso de falha
    set_new_handler( customNewHandler );

    // aloca memória para ptr[ i ]; customNewHandler será
    // chamado em caso de falha de memória
    for ( int i = 0; i < 100; i++ ) {
        ptr[ i ] = new double[ 5000000 ];
        cout << "Allocated 5000000 doubles in ptr["
            << i << "]!\n";
    }

    return 0;
}

```



## Hierarquia da Biblioteca Padrão de Exceção

- Hierarquia de exceção
  - Classe base de exceção (<exception>)
    - Função virtual `what`, sobrescrita para prover mensagens de erro
  - Classes derivadas
    - `runtime_error`, `logic_error`
    - `bad_alloc`, `bad_cast`, `bad_typeid`
      - Disparada por `new`, `dynamic_cast` e `typeid`
- Para pegar todas as exceções
  - `catch(...)`
  - `catch(exception AnyException)`
    - Não irá pegar exceções definidas por usuários que não foram derivadas da classe `exception`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Oitavo Exemplo usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 8
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>

using namespace std;

void myunexpected () {
    cerr << "unexpected called!\n";
    throw 0; // dispara int (na especificação de exceção)
}

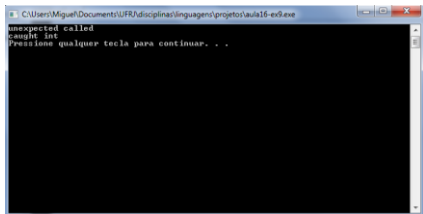
void myfunction () throw (int) {
    throw 'x'; // dispara char (não na especificação de exceção)
}

int main (void) {
    try {
        myunexpected();
    }
    catch (...) {
        cerr << "caught int!\n";
    }
    catch (...) {
        cerr << "caught other exception (non-compliant compiler)!\n";
    }

    return 0;
}
```

## Oitavo Exemplo usando Tratamento de Exceção em C++

```
/*
 * Aula 16 - Exemplo 8
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>
```



```
return 0;
}
```

## Exemplo 1

- Escreva um programa que dispare uma exceção caso haja uma tentativa de acesso a uma posição fora do escopo definido em um `vector`. Utilize um objeto da classe `out_of_range` para pegar a exceção.

?

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo 1

```
/*
 * Aula 16 - Exemplo 9
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <stdexcept>
#include <vector>

using namespace std;

int main (void) {
    vector<int> myvector(10);

    try {
        myvector.at(20) = 100; // vector dispara um tratamento
    }

    catch (out_of_range& oor) {
        cerr << "Out of Range error: " << oor.what() << endl;
    }

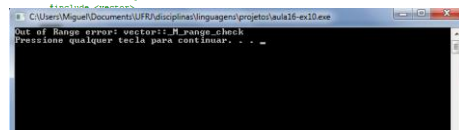
    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo 1

```
/*
 * Aula 16 - Exemplo 9
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <stdexcept>
#include <vector>
```



```
return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo 2

- Escreva um programa que dispare uma exceção não listada nos especificadores da exceção (lista de disparo) e redisparada usando o tipo `bad_exception`.



## Exemplo 2

```
/*
 * Aula 16 - Exemplo 10
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>

using namespace std;

void myunexpected () {
    cerr << "unexpected handler called\n";
    throw;
}

void myfunction () throw (int, bad_exception) {
    throw 14; // dispara obje (não está na especificação da exceção)
}

int main () {
    set_unexpected (myunexpected);
    try {
        myfunction();
    }
    catch (int) { cerr << "caught int\n"; }
    catch (bad_exception be) { cerr << "caught bad_exception\n"; }
    catch (...) {
        cerr << "caught other exception (non-compliant compiler)\n";
    }
    return 0;
}
```

## Exemplo 2

```
/*
 * Aula 16 - Exemplo 10
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>
```

```
try {
    myfunction();
}
catch (int) { cerr << "caught int\n"; }
catch (bad_exception be) { cerr << "caught bad_exception\n"; }
catch (...) {
    cerr << "caught other exception (non-compliant compiler)\n";
}
return 0;
}
```

## Exemplo 3

- Escreva um programa que dispare uma exceção se a idade do cadastro for menor que 18 anos. Para isso crie uma Classe Cadastro que recebe um nome e uma idade e utilize um objeto da Classe UnderAgeException para disparar uma exceção.



## Exemplo 3

```
/*
 * Aula 16 -- Exemplo 11
 * Arquivo cadastroCap16Ex12.h
 * Autor: Miguel Campista
 */
#include <iostream>
#include <string>

using namespace std;

class Cadastro {
public:
    Cadastro (string, int);
    void setName (string);
    void setAge (int);
    string getName () const;
    int getAge () const;
private:
    string name;
    int age;
};
```

## Exemplo 3

```
/*
 * Aula 16 -- Exemplo 11
 * Arquivo cadastroCap16Ex12.cpp
 * Autor: Miguel Campista
 */
#include "CadastroCap16Ex12.h"
#include "UnderAgeException.h"

Cadastro::Cadastro (string n, int a) : name (n) {
    age = 0;
    setAge (a);
}

void Cadastro::setName (string n) { name = n; }

void Cadastro::setAge (int a) {
    try {
        if (a < 18)
            throw UnderAgeException ();
        else
            age = a;
    }
    catch (UnderAgeException e) {
        cout << "\n*** Error: " << e.what () << " ***\n" << endl;
        cout << "*** Age not set for " << name << "!\n" << endl;
    }
}

string Cadastro::getName () const { return name; }

int Cadastro::getAge () const { return age; }
```

## Exemplo 3

```
/*
 * Aula 16 -- Exemplo 11
 * Arquivo underageexceptCap16Ex12.h
 * Autor: Miguel Campista
 */
#include <exception>

class UnderAgeException : public exception {
public:
    virtual const char * what () const throw () {
        return "Under Age";
    }
};
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo 3

```
/*
 * Aula 16 -- Exemplo 11
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "CadastroCap16Ex12.h"

int main() {
    Cadastro cad1 ("Fulano", 20);
    cout << "Name: " << cad1.getName () << "\nAge: " << cad1.getAge () << endl;

    Cadastro cad2 ("Cicrano", 17);
    cout << "Name: " << cad2.getName () << "\nAge: " << cad2.getAge () << endl;

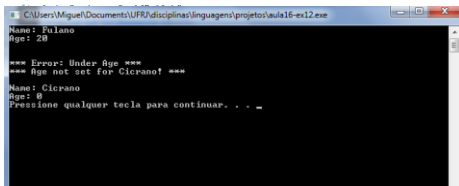
    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo 3

```
/*
 * Aula 16 -- Exemplo 11
 * Arquivo Principal
 * Autor: Miguel Campista
 */
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projeto\aula16-ex12.exe
Name: Fulano
Age: 20
*** Error: Under Age ***
*** Age not set for Cicrano! ***
Name: Cicrano
Age: 17
Pressione qualquer tecla para continuar. . . .
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Leitura Recomendada

- Capítulos 16 do livro
  - Deitel, "C++ How to Program", 5th edition, Editora Prentice Hall, 2005

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista