Programação Orientada a Objetos para Redes de Computadores

Prof. Miguel Elias Mitre Campista

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

PARTE 2

PROGRAMAÇÃO EM C++ - TRATAMENTO DE EXCEÇÃO

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

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

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
```

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!

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

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;`**

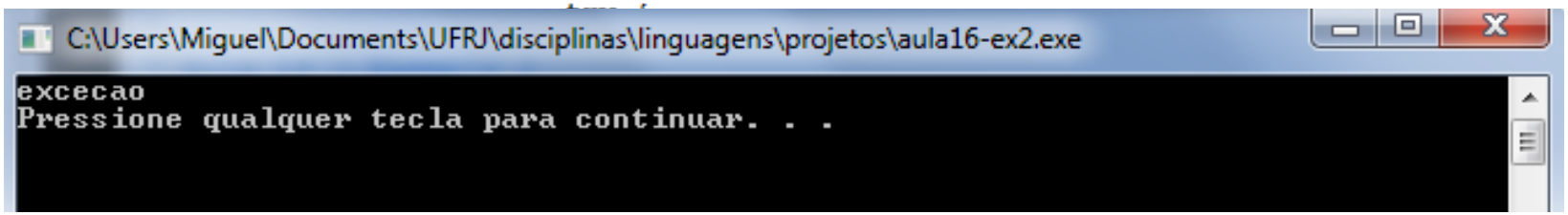
- Objetos de exceção
  - Classe base exceção (**`<exception>`**)
  - Construtor pode receber uma string (para descrever a exceção)
    - Função membro **`what()`** retorna essa string

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;
}
```

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;
}
```

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`**

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";
            }
};
```

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" ) {}
};
```

Primeiro Exemplo Usando Tratamento de Exceção em C++

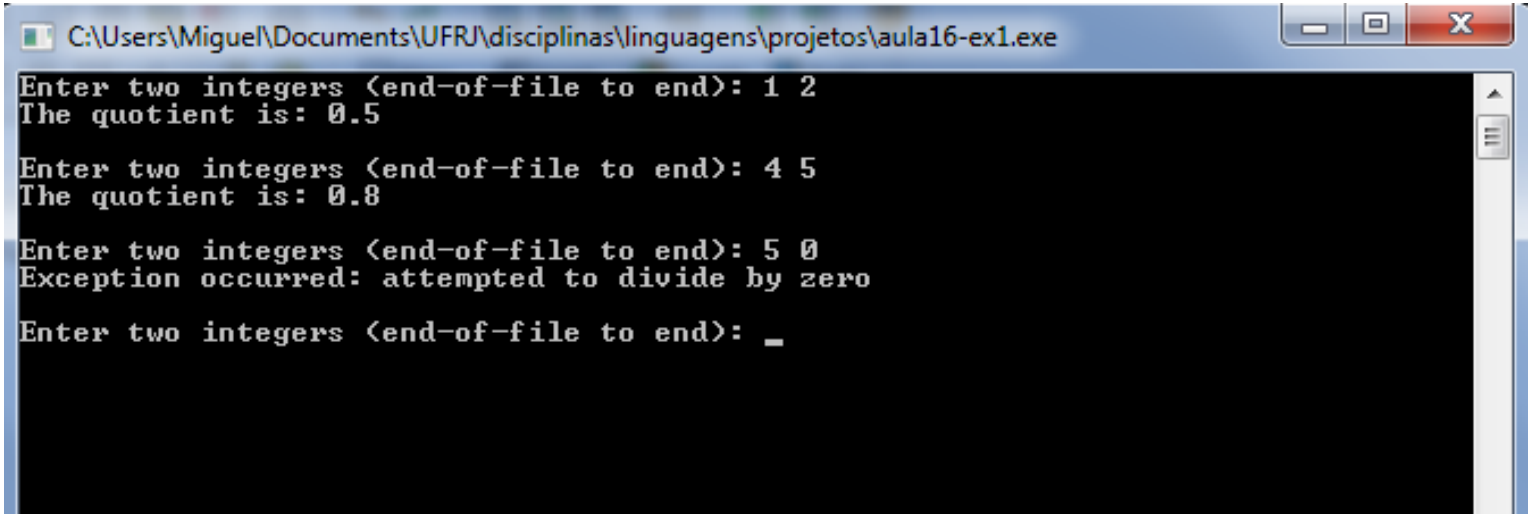
```
/*
 * Aula 16 - Exemplo 1
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "erroCap16Ex1.h"

// 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;
}
```

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
            // e 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;
}
```

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
```




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

        return 0;
}
```

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

Redisparo de uma Exceção (Rethrow exception)

- Para redisparar
  - Usado com o sentença "`throw;`"
    - Sem argumentos
    - Termina uma função

Segundo Exemplo Usando Tratamento de Exceção em C++

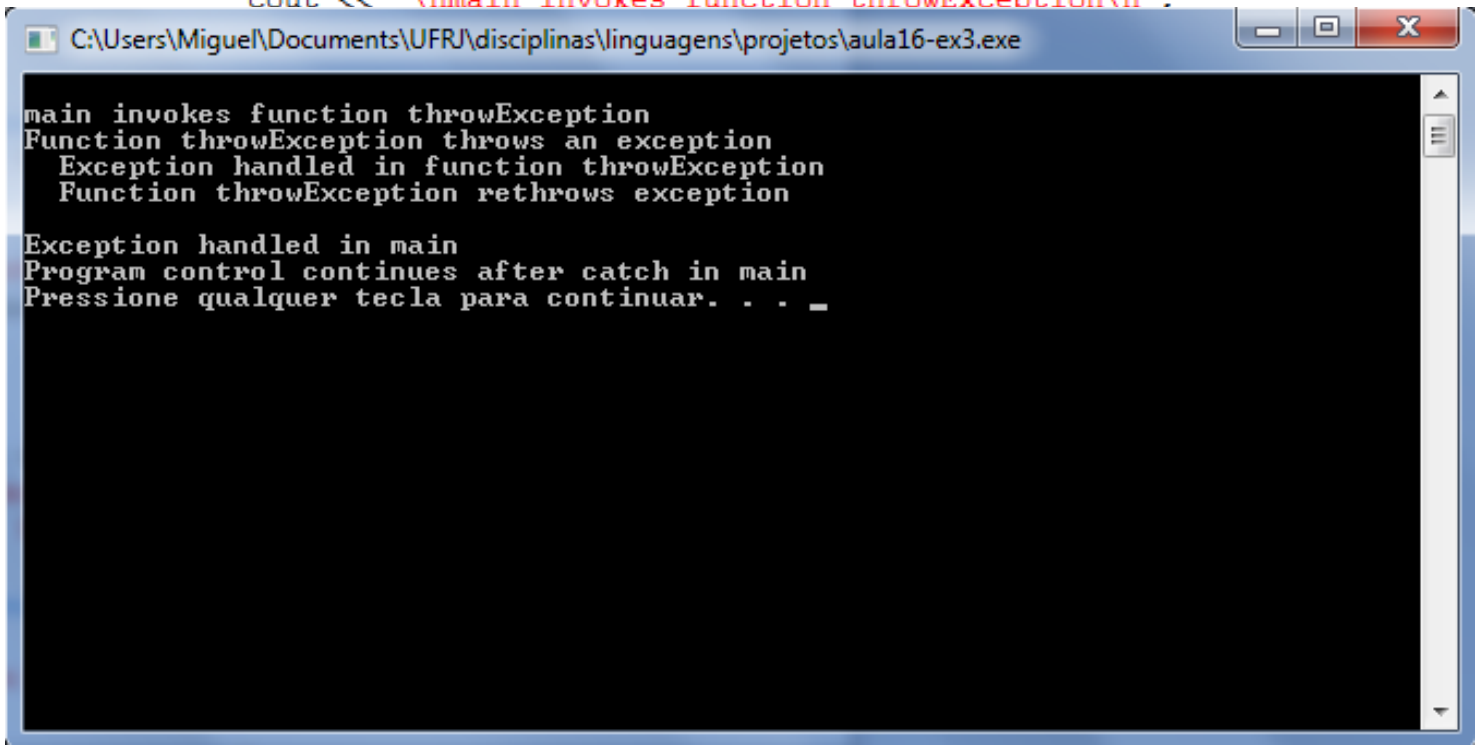
```
/*
 * Aula 16 - Exemplo 2
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>

using namespace std;

// dispara, trata e redispara 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;  // redispara a exceção para processamento posterior
    }
    cout << "This also should not print\n";
}
```

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 << "\n\nException handled in main\n";
    }

    cout << "Program control continues after catch in main\n";

    return 0;
}
```

Segundo Exemplo Usando Tratamento de Exceção em C++

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



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

Especificação de Exceções

- Tipo de exceção disparada por uma função pode ser limitada
  - Adição de sufixo no protótipo da função

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

  - 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

```
// 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

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

Processamento de Exceções Unexpected

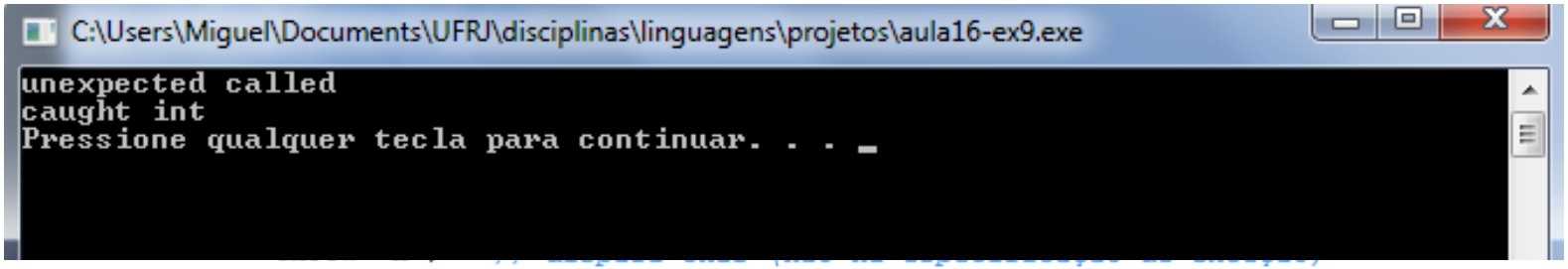
- Argumentos para as funções de definição: **`set_unexpected`** e **`set_terminate`**
  - Recebe ponteiro para função
    - Função não deve receber argumentos
  - Retorna `void`

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)
}

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)
}

int main () {
          myfunction();
     }
     catch (int) { cerr << "caught int\n"; }

     return 0;
}
```

E se inserirmos o tipo char na *throw list*?

Segundo Exemplo Usando Tratamento de Exceção em C++

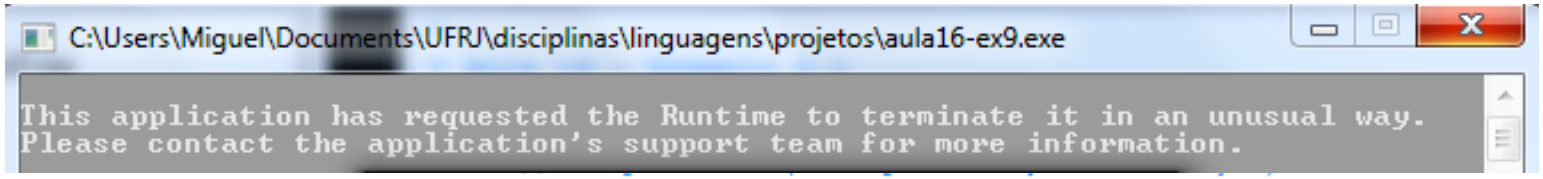
```
/*
 * Aula 16 - Exemplo 2.1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <exception>

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 (não 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 (não na especificação de exceção)
}
```

Por que não?

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

    return 0;
}
```

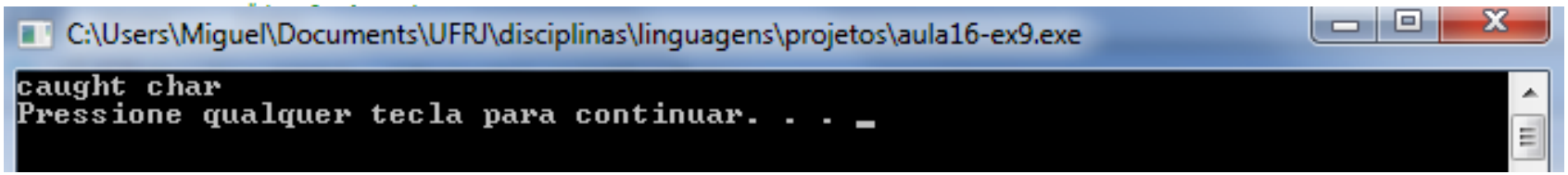
Segundo Exemplo Usando Tratamento de Exceção em C++

Faltava definir o bloco `catch` correspondente?

```
/*
 * Aula 16 - Exemplo 2.1
 */
#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 (não 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
 */
```




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

void myfunction () throw (int, char) {
     throw 'x';    // dispara char (não 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 'y';      // dispara cha
}

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++

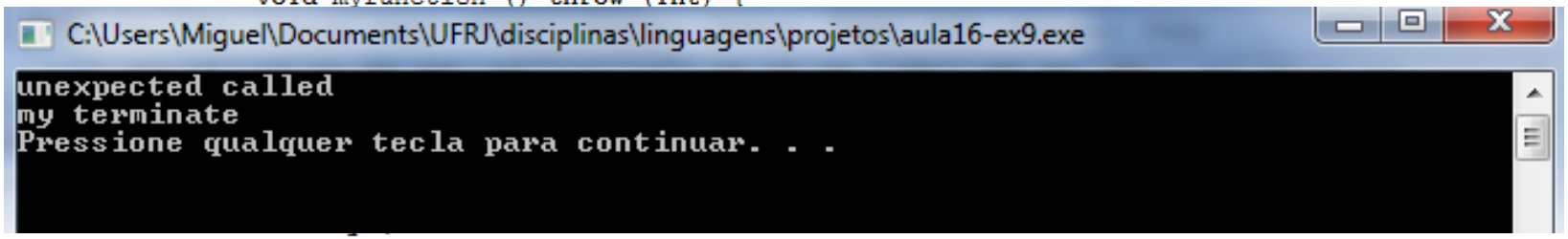
```
/*
 * Aula 16 - Exemplo 2.1
 * Arquivo Principal
 * Autor: Miguel Campista
 */

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

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

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

void myfunction () throw (int) {
```

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




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

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

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

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

    return 0;
}
```

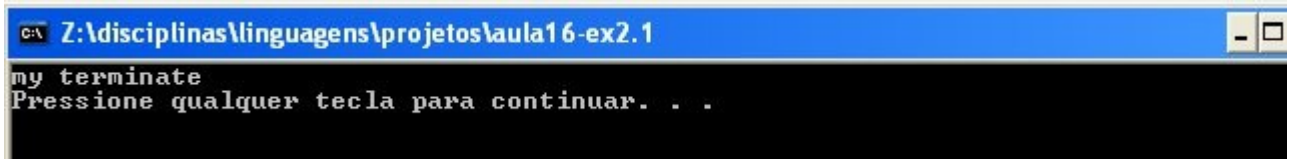
Segundo Exemplo Usando Tratamento de Exceção em C++

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

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

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

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

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

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

    return 0;
}
```

Tipo `char` é inserido na lista de disparo. Nesse caso, o que acontece?

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 'y';
}

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

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

    return 0;
}
```



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;

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

// função2 invoca função3
void function2() throw ( runtime_error ) {
     function3(); // 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 execução das funções é terminada...

```
using namespace std;

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

// função2 invoca função3
void function2() throw ( runtime_error ) {
     function3(); // 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 função1
      try {
            function1(); // primeiro
      }

      // trata erro run-time
      catch ( runtime_error &error ) { // quinto
            cout << "Exception occurred: " << error.what() << endl;
      }

      return 0;
}
```

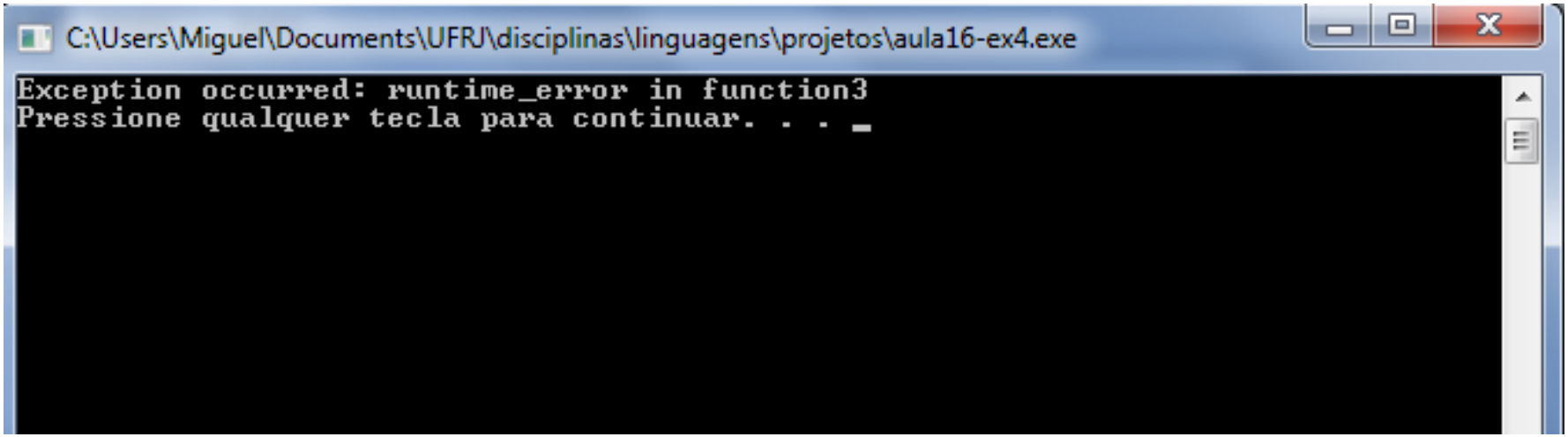
Terceiro Exemplo usando Tratamento de Exceção em C++

```
// demonstra liberação de pilha
int main() {
// invoca função1
      try {
            function1(); // primeiro
      }

      // trata erro run-time
      catch ( runtime_error &error ) { // quinto
            cout << "Exception occurred: " << error.what() << endl;
      }

      return 0;
}
```

Exceção só é pega na função principal...

Terceiro Exemplo usando Tratamento de Exceção em C++

```
// demonstra liberação de pilha
int main() {
// invoca função1
     try {
          function1(); // primeiro
     }
```



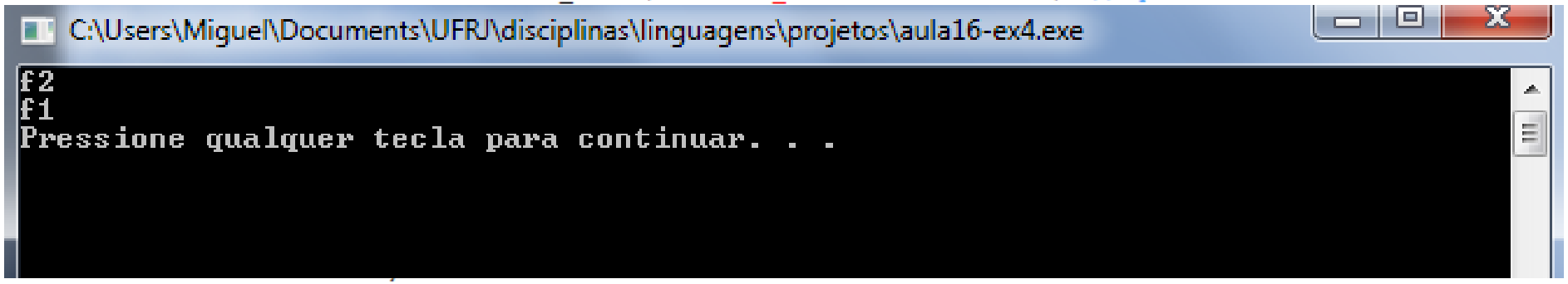
Terceiro Exemplo usando Tratamento de Exceção em C++

```
/*
 * 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" ); // quarto
}

// função2 invoca função3
void function2() throw ( runtime_error ) {
     try {
         function3();
     } // terceiro
     catch (runtime_error &e) {
           cout << "f2" << endl;
     }
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
     function2(); // segundo
     cout << "f1\n";
}
```

E se fosse assim?

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" ); // quarto
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
     function2(); // segundo
     cout << "f1\n";
}
```



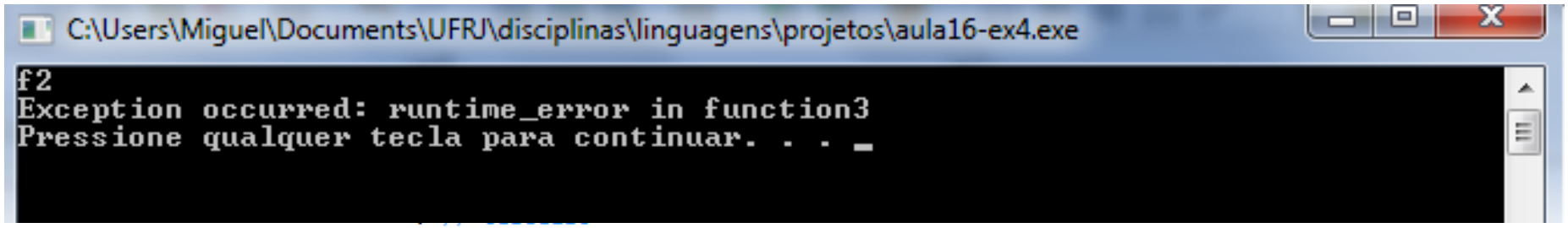
Terceiro Exemplo usando Tratamento de Exceção em C++

```
/*
 * 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" ); // quarto
}

// função2 invoca função3
void function2() throw ( runtime_error ) {
     try {
         function3();
     } // terceiro
     catch (runtime_error &e) {
           cout << "f2" << endl;
           throw;
     }
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
     function2(); // segundo
     cout << "f1\n";
}
```

E se fosse assim?

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 ) {
```




```
    catch (runtime_error &e) {
        cout << "f2" << endl;
        throw;
    }
}

// função1 invoca função2
void function1() throw ( runtime_error ) {
    function2(); // segundo
    cout << "f1\n";
}
```

Construtores, Destrutores e Tratamento de Exceção

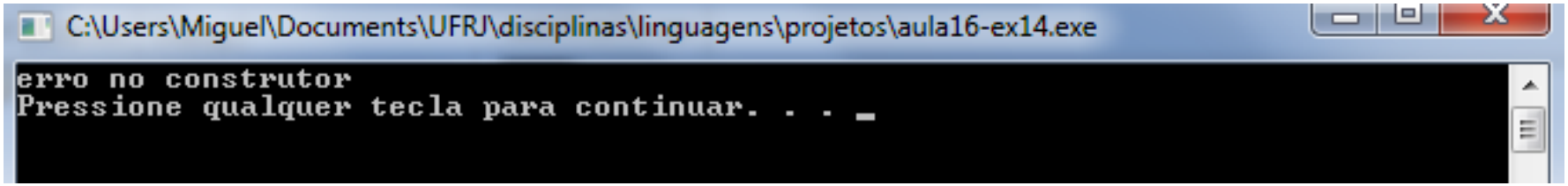
- Erro no construtor
  - `new` falha
    - Por exemplo: não pode alocar memória
  - Contrutor não pode retornar um valor: **Como informar o usuário?**
    - Espera-se que o usuário examine o objeto e note os erros?
    - Uso de variáveis globais?
  - **Boa alternativa: disparar uma exceção**
    - Liberação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




```
    }
    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

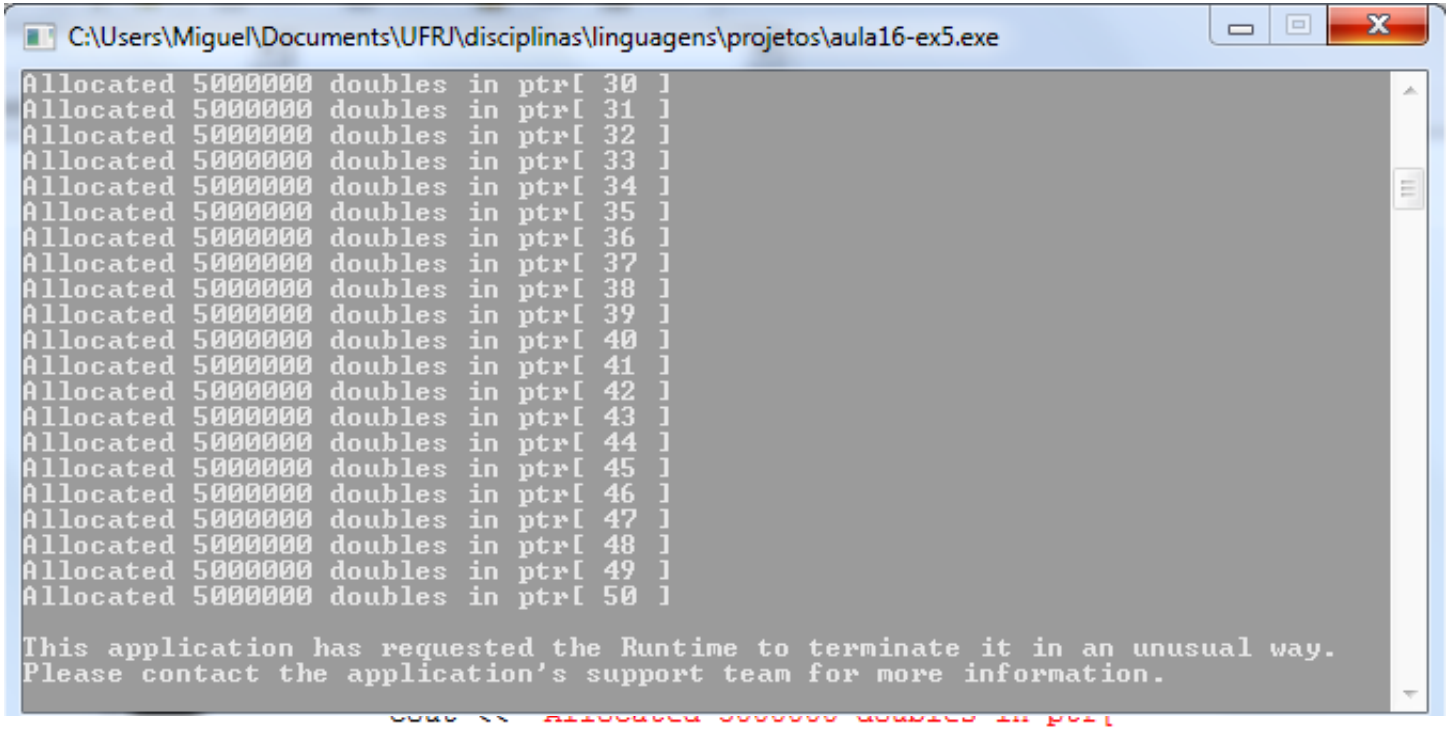
- Quando o **new** falha para alocar memória...
  - Deve-se disparar exceção do tipo **bad_alloc**
    - Definida em **<new>**
  - Alguns compiladores têm **new** retornando 0 (zero)
  - Resultado depende do compilador

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 ];

          // new retorna 0 em caso de falha para alocar memória
          if ( ptr[ i ] == 0 ) {
               cout << "Memory allocation failed for ptr[ "
                    << i << " ]\n";
               break;
          } else
               // alocação bem sucedida 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
```



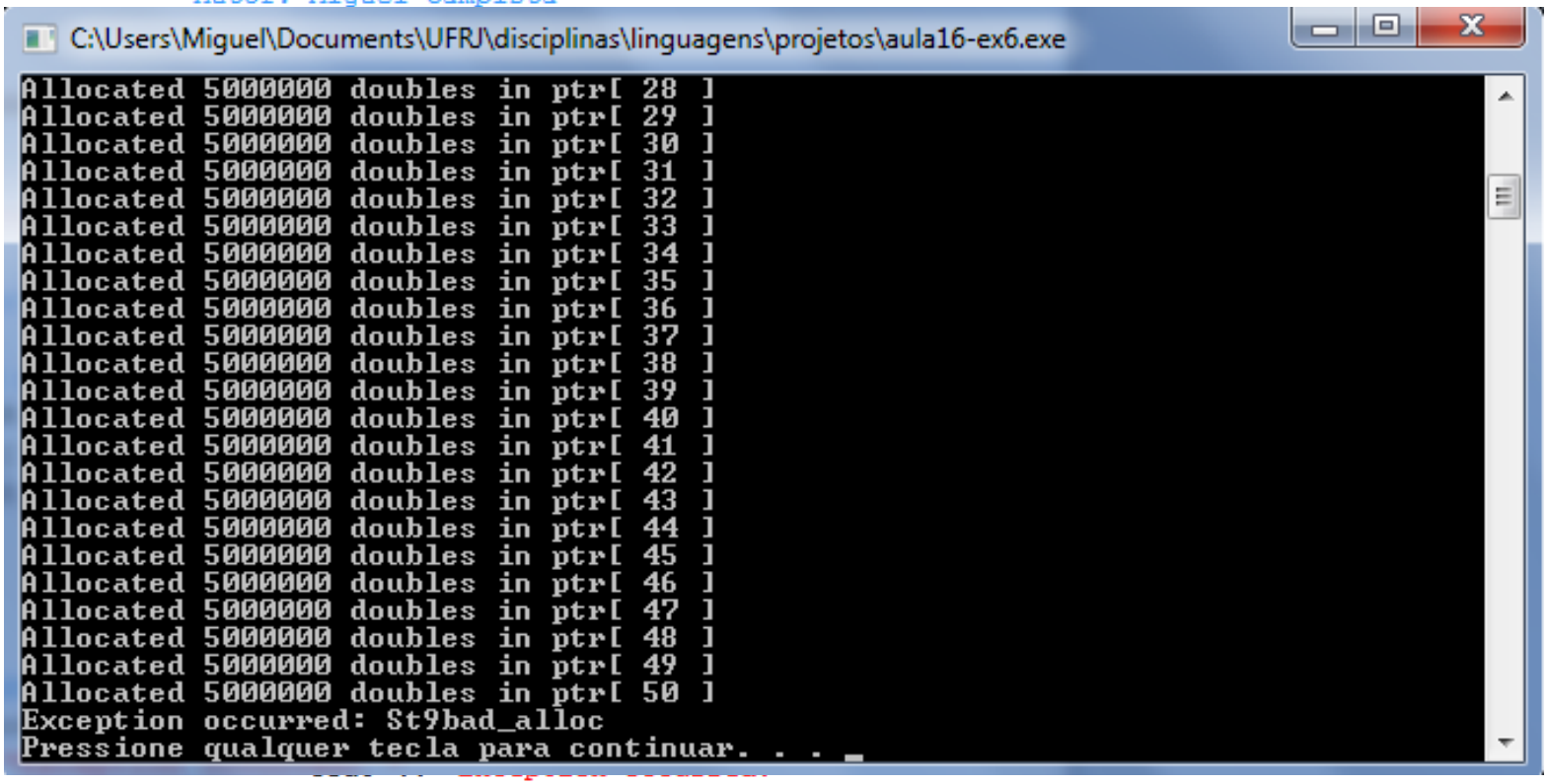

```
                << i << " ]\n";
        }

        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 ];

    // tenta alocar memória
    try {
          // aloca memória para ptr; new dispara 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
```




```
                << memoryAllocationException.what() << endl;
        }

        return 0;
}
```

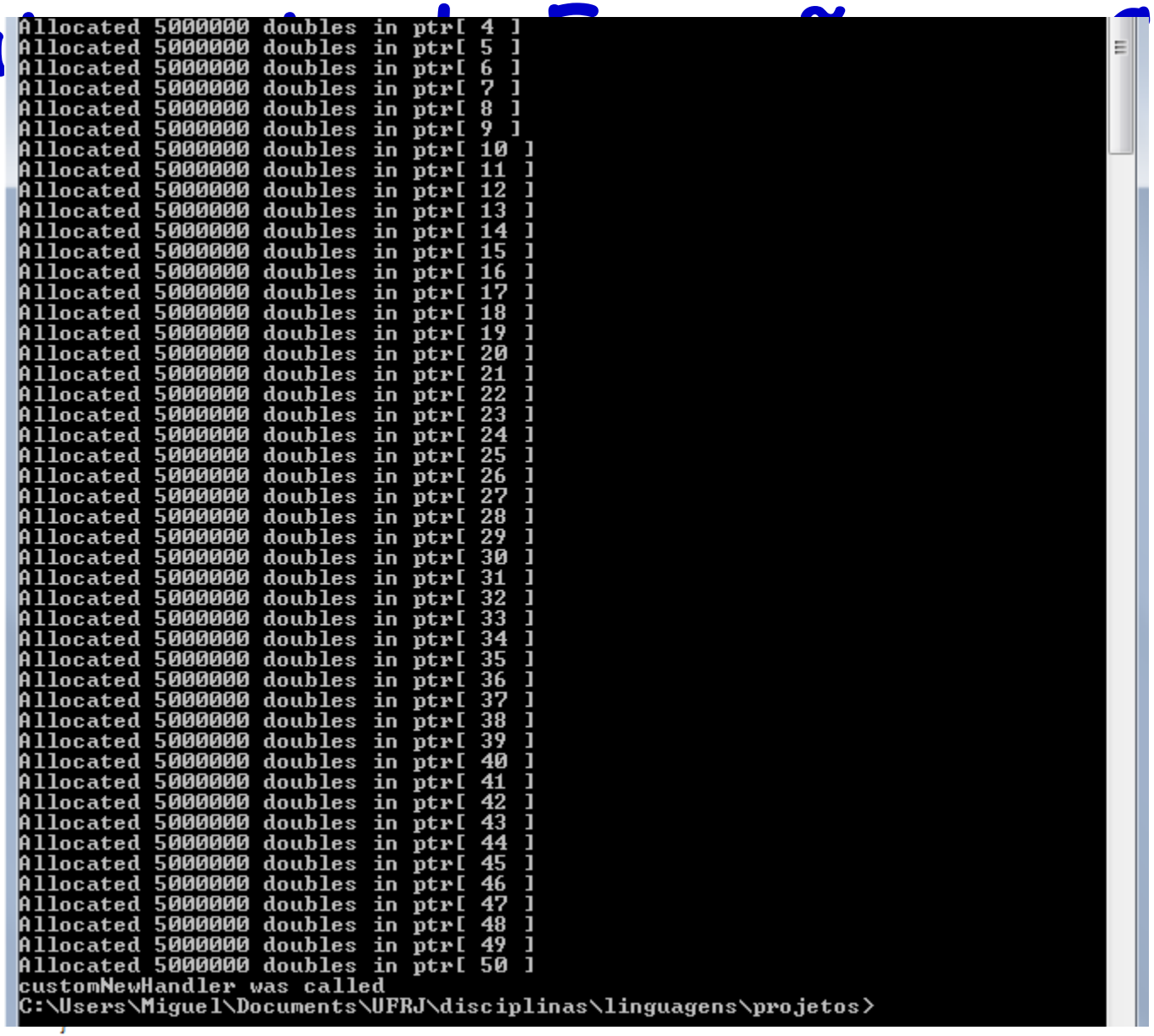
Processamento de Novas Falhas

- **`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() {
      cerr << "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
      // de alocação de memória
      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;
}
```

Sexto Exemplo usando Tra...++



Classe auto_ptr e Alocação Dinâmica de Memória

- Declarar ponteiro, alocar memória com **`new`**
  - E se a memória for alocada corretamente, mas a exceção ocorrer antes de liberar (**`delete`**) o objeto?
    - Vazamento de memória

Classe auto_ptr e Alocação Dinâmica de Memória

- Classe template **auto_ptr**
  - Cabeçalho **<memory>**
  - Quando ponteiro sai do escopo, chama-se **delete**
    - Previne vazamento de memória
  - Sobrecarrega ponteiros regulares (* e ->)
    - Objeto **auto_ptr** pode ser usado como um ponteiro

```
auto_ptr< MyClass > newPointer( new MyClass() );
\\ newPointer aponta para objeto alocado dinamicamente
```

Sétimo Exemplo usando Tratamento de Exceção em C++

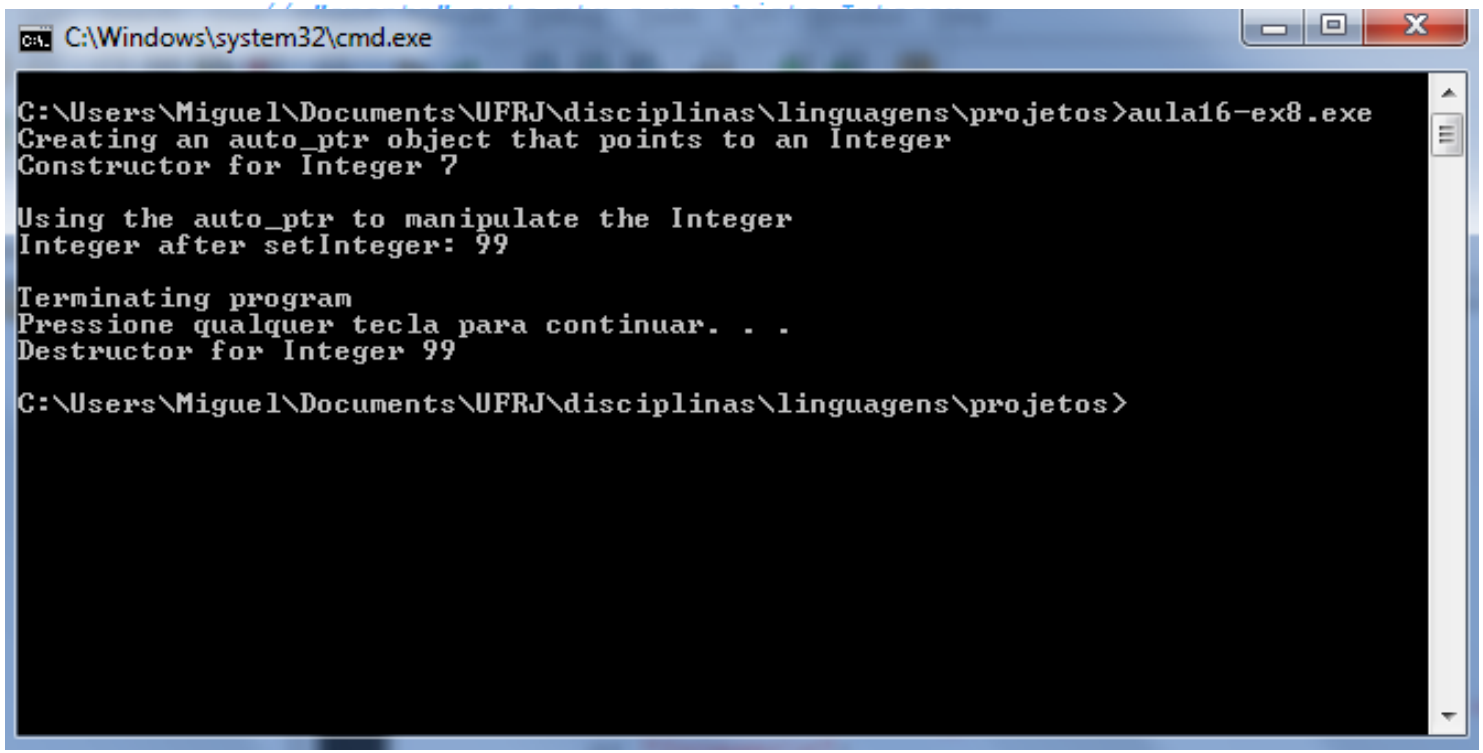
```
/*
 * Aula 16 - Exemplo 7
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <memory>

using namespace std;

class Integer {
      public:
            // Construtor de Integer constructor
            Integer( int i = 0 ) : value( i ) {
                  cout << "Constructor for Integer " << value << endl;
            }
            // Destrutor de Integer
            ~Integer() {
                  cout << "Destructor for Integer " << value << endl;
            }
            // Função para atribuir Integer
            void setInteger( int i ) { value = i; }
            // Função para retornar Integer
            int getInteger() const { return value; }
      private:
            int value;
};
```

Sétimo Exemplo usando Tratamento de Exceção em C++

```
int main() {
      cout << "Creating an auto_ptr object that points to an "
           << "Integer\n";
      // "aponta" auto_ptr a um objeto Integer
      auto_ptr< Integer > ptrToInteger( new Integer( 7 ) );

      cout << "\nUsing the auto_ptr to manipulate the Integer\n";

      // use auto_ptr para atribuir valor a Integer
      ptrToInteger->setInteger( 99 );

      // use auto_ptr para obter o valor Integer
      cout << "Integer after setInteger: "
           << ( *ptrToInteger ).getInteger()
           << "\n\nTerminating program" << endl;

      return 0;
}
```

Sétimo Exemplo usando Tratamento de Exceção em C++

```
int main() {
      cout << "Creating an auto_ptr object that points to an "
           << "Integer\n";
```



Hierarquia da Biblioteca Padrão de Exceção

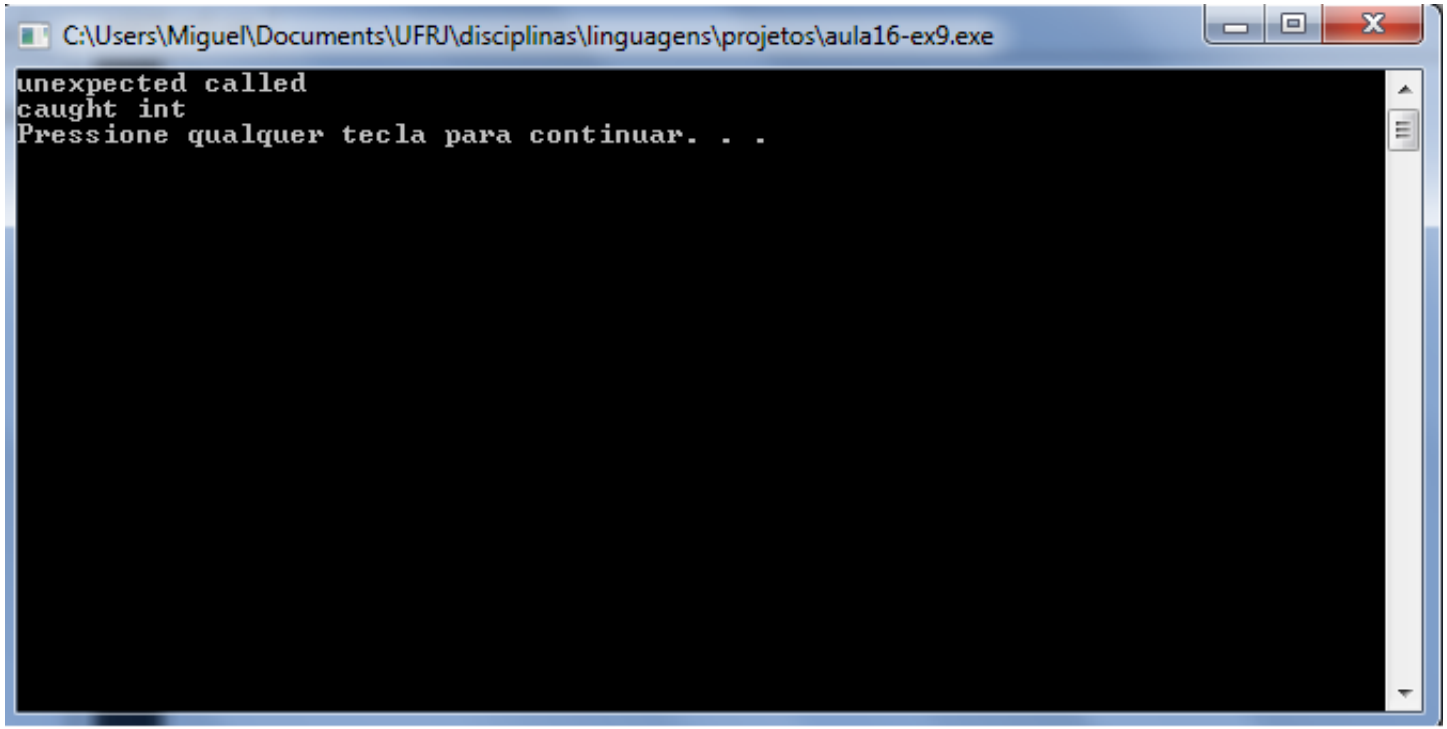
- Hierarquia de exceção
  - Classe base de exceção (**<exception>**)
    - Função virtual **what**, sobrescrita para prover mensagens de erro
  - Classes derivadas
    - **runtime_error**, **logic_error**
    - **bad_alloc**, **bad_cast**, **bad_typeid**
      - Disparada por **new**, **dynamic_cast** 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

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) {
    set_unexpected (myunexpected);
    try {
         myfunction();
    }
    catch (int) { 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.

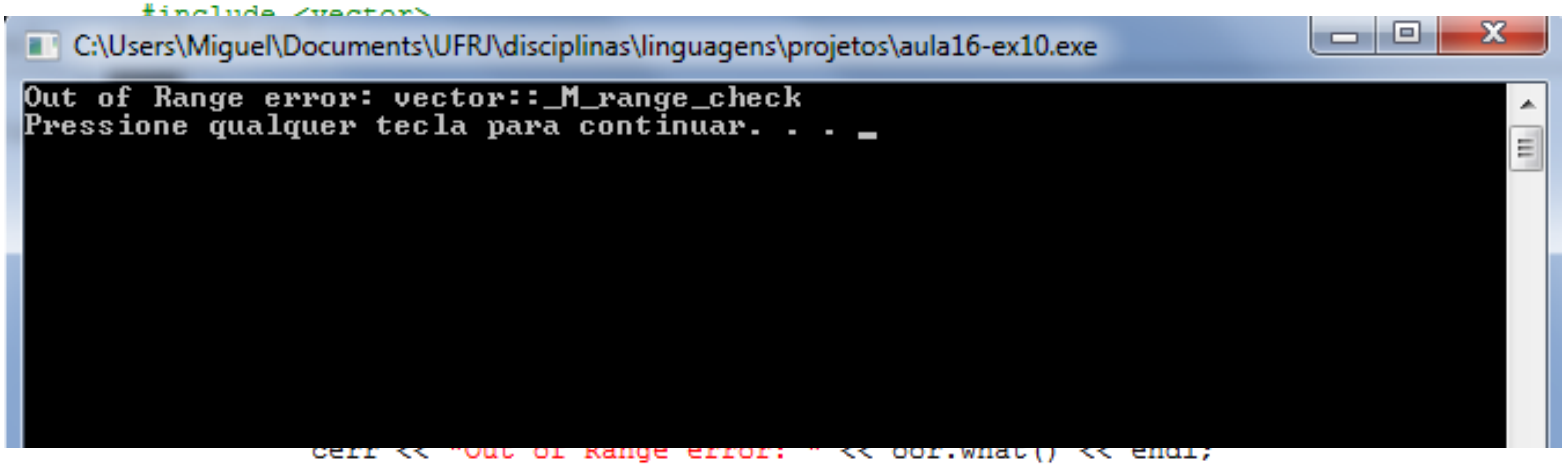
?

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;
}
```

Exemplo 1

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




```
        cerr << "Out of Range error: " << oor.what() << endl;
    }

    return 0;
}
```

Exemplo 2

- 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 2

```
/*
 * 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 2

```
/*
 * Aula 16 -- Exemplo 11
 * Arquivo cadastroCap16Ex12.cpp
 * Autor: Miguel Campista
*/
#include "cadastroCap16Ex12.h"
#include "underageexceptCap16Ex12.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\n*** Error: " << e.what () << " ***" << endl;
           cout << "*** Age not set for " << name << "! ***\n" << endl;
     }
}

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

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

Exemplo 2

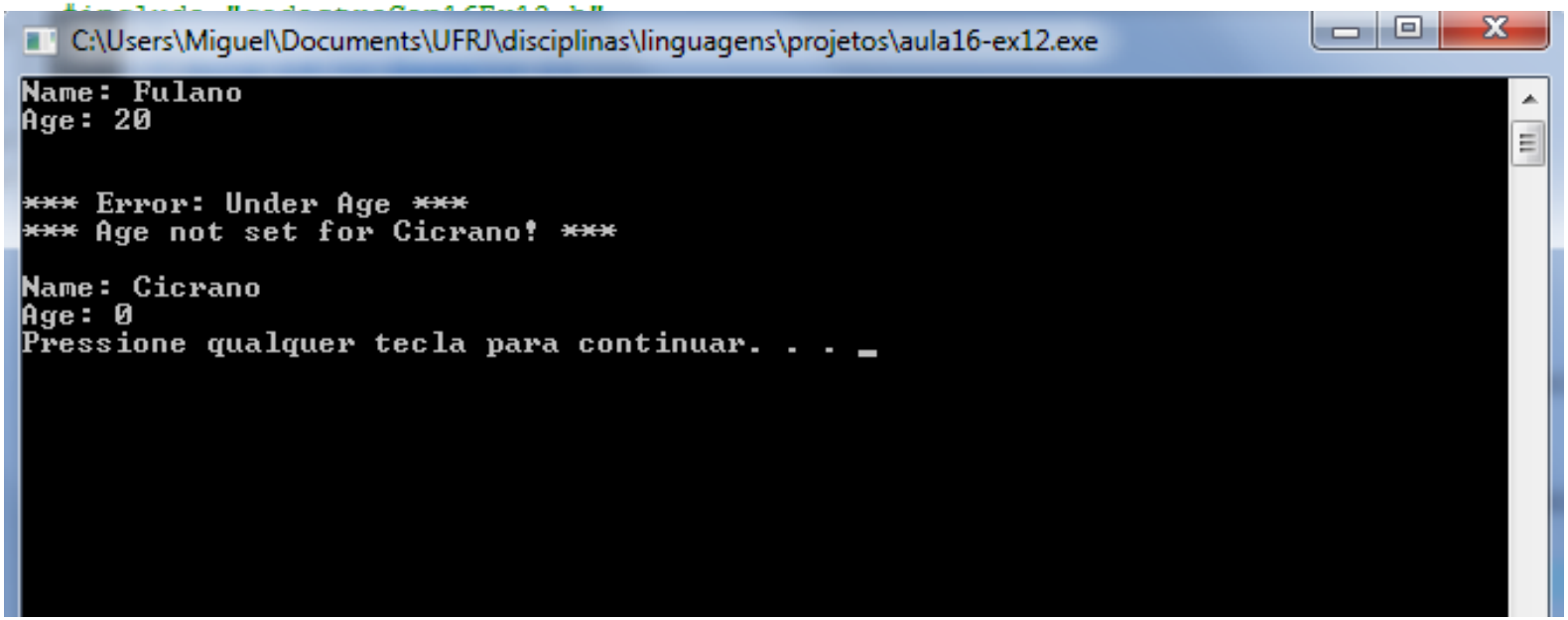
```
/*
 * Aula 16 -- Exemplo 11
 * Arquivo underageexceptCap16Ex12.h
 * Autor: Miguel Campista
*/
#include <exception>

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

Exemplo 2

```
/*
 * 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;
}
```

Exemplo 2

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



Leitura Recomendada

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