C++
Básico ao Avançado
Todos, exceto alguns

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

Vamos fazer umas diviões

Vamos fazer umas diviões

- Vocês já dividiram por zero hoje?

Vamos fazer umas diviões

- Vocês já dividiram por zero hoje?
- O que acontece se a gente divide por zero?

Vamos fazer umas diviões

- Vocês já dividiram por zero hoje?
- O que acontece se a gente divide por zero?
- E no programa, o que acontece?

Vamos fazer umas diviões

- Vocês já dividiram por zero hoje?
- O que acontece se a gente divide por zero?
- E no programa, o que acontece?
- Já tentaram imprimir o que sai?

Vamos fazer umas diviões

- Vocês já dividiram por zero hoje?
- O que acontece se a gente divide por zero?
- E no programa, o que acontece?
- Já tentaram imprimir o que sai?

```
double divide(double A, double B)
{
  return A/B;
}
//...
double K = divide(2,0);
double W = divide(1,K);
```

- *Ah, mas eu resolvi problemas de divisão por 0 com um* if

- *Ah, mas eu resolvi problemas de divisão por 0 com um* `if`
- Ok, mas esse é o único erro que acontece?

- *Ah, mas eu resolvi problemas de divisão por 0 com um* `if`
- Ok, mas esse é o único erro que acontece?
- Vamos lembrar dos momentos do programa:

- *Ah, mas eu resolvi problemas de divisão por 0 com um* `if`
- Ok, mas esse é o único erro que acontece?
- Vamos lembrar dos momentos do programa:
  - Tempo de compilação

- *Ah, mas eu resolvi problemas de divisão por 0 com um* `if`
- Ok, mas esse é o único erro que acontece?
- Vamos lembrar dos momentos do programa:
  - Tempo de compilação
  - Tempo de execução

- *Ah, mas eu resolvi problemas de divisão por 0 com um* `if`
- Ok, mas esse é o único erro que acontece?
- Vamos lembrar dos momentos do programa:
  - Tempo de compilação
  - Tempo de execução
- Mas isso evita o erro, não o corrige ou trata

- *Ah, mas eu resolvi problemas de divisão por 0 com um* `if`
- Ok, mas esse é o único erro que acontece?
- Vamos lembrar dos momentos do programa:
  - Tempo de compilação
  - Tempo de execução
- Mas isso evita o erro, não o corrige ou trata
- Toda vez que um erro acontece é lançada uma exceção

- *Ah, mas eu resolvi problemas de divisão por 0 com um* `if`
- Ok, mas esse é o único erro que acontece?
- Vamos lembrar dos momentos do programa:
  - Tempo de compilação
  - Tempo de execução
- Mas isso evita o erro, não o corrige ou trata
- Toda vez que um erro acontece é lançada uma exceção
- Exceções devem ser capturadas e tratadas

- *Ah, mas eu resolvi problemas de divisão por 0 com um* `if`
- Ok, mas esse é o único erro que acontece?
- Vamos lembrar dos momentos do programa:
  - Tempo de compilação
  - Tempo de execução
- Mas isso evita o erro, não o corrige ou trata
- Toda vez que um erro acontece é lançada uma exceção
- Exceções devem ser capturadas e tratadas
- Para isso, existe um controle de fluxo especial que coleta exceções e permite tratamento

- Primeiro precisamos de uma estrutura que comporte comandos que possam gerar erros

- Primeiro precisamos de uma estrutura que comporte comandos que possam gerar erros
- Esta estrutura utiliza as palavras-chave `try` e `catch`

- Primeiro precisamos de uma estrutura que comporte comandos que possam gerar erros
- Esta estrutura utiliza as palavras-chave `try` e `catch`
- O `try` contém o bloco onde as exceções podem ser lançadas

- Primeiro precisamos de uma estrutura que comporte comandos que possam gerar erros
- Esta estrutura utiliza as palavras-chave `try` e `catch`
- O `try` contém o bloco onde as exceções podem ser lançadas
- O `catch` recebe como argumento a declaração de um objeto classe de exceção

- Primeiro precisamos de uma estrutura que comporte comandos que possam gerar erros
- Esta estrutura utiliza as palavras-chave `try` e `catch`
- O `try` contém o bloco onde as exceções podem ser lançadas
- O `catch` recebe como argumento a declaração de um objeto classe de exceção, além de conter o boco de tratamento

- Primeiro precisamos de uma estrutura que comporte comandos que possam gerar erros
- Esta estrutura utiliza as palavras-chave `try` e `catch`
- O `try` contém o bloco onde as exceções podem ser lançadas
- O `catch` recebe como argumento a declaração de um objeto classe de exceção, além de conter o boco de tratamento
- É possível que um bloco lance mais de uma exceção

- Primeiro precisamos de uma estrutura que comporte comandos que possam gerar erros
- Esta estrutura utiliza as palavras-chave `try` e `catch`
- O `try` contém o bloco onde as exceções podem ser lançadas
- O `catch` recebe como argumento a declaração de um objeto classe de exceção, além de conter o boco de tratamento
- É possível que um bloco lance mais de uma exceção, então é possível utilizar mais de um `catch`

Sintaxe

Sintaxe

```
//...
try
{
  //...
}
catch(<exceção1> <nome1>)
{
  //...
}
//...
catch(esxceçãoN> <nomeN>)
{
  //...
}
//...
```

Lançando exceções

Lançando exceções

- É possível lançar exceções próprias

Lançando exceções

- É possível lançar exceções próprias
- O caso da divisão por zero é um bom exemplo

Lançando exceções

- É possível lançar exceções próprias
- O caso da divisão por zero é um bom exemplo, não é uma exceção padrão

Lançando exceções

- É possível lançar exceções próprias
- O caso da divisão por zero é um bom exemplo, não é uma exceção padrão
- Para lançar uma exceção utiliza-se a palavra-chave `throw`

Lançando exceções

- É possível lançar exceções próprias
- O caso da divisão por zero é um bom exemplo, não é uma exceção padrão
- Para lançar uma exceção utiliza-se a palavra-chave `throw`
- É semelhante ao `return`

Lançando exceções

- É possível lançar exceções próprias
- O caso da divisão por zero é um bom exemplo, não é uma exceção padrão
- Para lançar uma exceção utiliza-se a palavra-chave `throw`
- É semelhante ao `return`, mas o *retorno* é capturado pelo `try`

Lançando exceções

- É possível lançar exceções próprias
- O caso da divisão por zero é um bom exemplo, não é uma exceção padrão
- Para lançar uma exceção utiliza-se a palavra-chave `throw`
- É semelhante ao `return`, mas o *retorno* é capturado pelo `try`
- O item lançado é um objeto

Lançando exceções

- É possível lançar exceções próprias
- O caso da divisão por zero é um bom exemplo, não é uma exceção padrão
- Para lançar uma exceção utiliza-se a palavra-chave `throw`
- É semelhante ao `return`, mas o *retorno* é capturado pelo `try`
- O item lançado é um objeto, herdeiro da classe `exception`

Lançando exceções

- É possível lançar exceções próprias
- O caso da divisão por zero é um bom exemplo, não é uma exceção padrão
- Para lançar uma exceção utiliza-se a palavra-chave `throw`
- É semelhante ao `return`, mas o *retorno* é capturado pelo `try`
- O item lançado é um objeto, herdeiro da classe `exception`

```
throw <exceção>;
```

Contextualizando

Contextualizando

```
#include <stdexcept>
//...
class divZero: public
    runtime_error
{
public:
  divZero(): runtime_error("
      Dividiu por zero")
  {}
}
//...
double divide(double A,
    double B)
{
  if(B==0)
  {
    throw divZero();
  }
  return A/B;
}
//...
try
{
  double K = divide(2,0);
  double W = divide(1,K);
}
catch(divZero e)
{
  //Exibe e.what(), saída: "
      Erro: Dividiu por zero"
}
//...
```

Vamos testar!