INF 112: Programação II

Aula 20

- **Tratamento de exceções em C++**

Fábio R. Cerqueira, UFV, DPI, frcerqueira@gmail.com

Tratamento de exceções

➔ Apresentaremos agora uma introdução ao *tratamento de exceções*, que permite gerenciar os erros (e situações inesperadas) de uma forma organizada no programa.

➔ A ideia é invocar rotinas de tratamento de exceções quando estas ocorrerem no programa, separando assim a lógica do que se quer executar do tratamento de situações adversas.

➔ O tratamento de exceções fornece, portanto, um meio de transferir controle e informações de um ponto na execução de um programa para um "tratador de exceções" (*exception handler*).

Tratamento de exceções

➔ Quanto maior o número de situações de exceção que um determinado programa consegue lidar, mais robusto é este programa. Vejamos três abordagens bastante comuns para tratamento de erro:

1. *Ignorar* – neste caso a ação será a padrão do sistema, provavelmente a interrupção do programa, ou no mínimo do comando onde ocorreu o erro, e a exibição de uma mensagem ininteligível na saída padrão, gerada pelo sistema. Esta é abordagem mais simples, mas certamente não é a mais interessante, principalmente do ponto de vista do usuário final.

Tratamento de exceções

2. *Retornar o código de erro* – podendo ser por meio de variáveis globais, parâmetros de saída, ou retorno de funções. Este método aumenta a complexidade dos comandos e funções (mistura da lógica do programa com o tratamento de erros) e pode levar aos mesmos problemas da abordagem anterior, uma vez que o programador pode esquecer de testar os códigos de retorno. Esta é uma abordagem muito comum entre os programadores de C e Pascal e é ainda usada pelos programadores de C++.

Tratamento de exceções

- Um exemplo da abordagem 2:

```
struct Racional {
  int n,d;
};

Racional criaRacional(int num, int den, int &codRet)
{
  Racional nr = {num,den};
  if ( den == 0 ) codRet = 1;
  else codRet = 0;
  return nr;
}
```

Tratamento de exceções

3. *Utilizar tratadores de exceções* – linguagens mais modernas como C++ e Java disponibilizam recursos que permitem um melhor tratamento de exceções. Um tratador de exceção é uma parte do código que tem por objetivo recuperar a execução do programa após a ocorrência da exceção, permitindo que o sistema se comporte "suavemente", mesmo sob condições adversas. A sintaxe e a semântica dos tratadores de exceção têm tendido a uma certa uniformização nas linguagens mais modernas.

Tratamento de exceções

- O mecanismo de tratamento de exceções requer o uso de três palavras chave: `try`, `catch` e `throw`.

- Nos termos mais gerais, os comandos do programa que devem ser monitorados para as exceções estão contidos em um bloco de prova (`try`).

- Se uma exceção (ou erro) ocorrer dentro do bloco de prova, a mesma será disparada usando `throw`. A exceção é “pega” e processada por um bloco `catch`.

- Portanto, o bloco `catch` possui os comandos para tratamento da exceção.

Tratamento de exceções

- Segue o formato geral para tratamento de exceções em C++:

```
try {
// Comandos passíveis de gerar exceção.
// Em algum ponto aqui, ou em alguma função chamada
// a partir daqui, haverá a execução de um throw.
// Após o throw, o fluxo de execução será desviado
// para um catch (supondo que há um catch apropriado).
...
} catch(<classe da exceção 1> [variável]) {
// Comandos para tratamento da exceção 1
...
} catch(<classe da exceção 2> [variável]) {
// Comandos para tratamento da exceção 2
...
} catch(<classe da exceção n> [variável]) {
// Comandos para tratamento da exceção n
...
}
```

Tratamento de exceções

- Assim, nota-se que pode haver mais de um comando `catch` associado a um `try`.

- Se o bloco `catch` não possuir nenhum comando de `return` ou `exit`, então o programa é reassumido após o último bloco `catch`.

- Veja então que as instruções seguintes ao trecho onde ocorreu a exceção (instruções após o `throw`) não serão executadas.

- Note que somente um bloco `catch` será executado, sendo os outros ignorados. O bloco escolhido dependerá do tipo do objeto lançado pelo `throw`. Este tem o seguinte formado: `throw <classe>(<parâmetros>)` onde `<classe>` pode denotar uma classe pré definida pelo sistema ou uma classe qualquer definida pelo programador e `<parâmetros>` são os parâmetros do construtor da classe, caso os tenha. Veja que uma instância de um tipo é criada e lançada no momento do `throw`.

Tratamento de exceções

➔ O exemplo a seguir mostra um trecho de código em C++ com a geração e tratamento de exceção. Note que na geração da exceção um valor inteiro é lançado. O valor é capturado pela variável `e`. Em seguida, no `catch`, o valor recebido é exibido no console.

```
main() {
  ...
  try {
    ...
    throw 10;
    ... // Aqui não é executado
  } catch (int e) {
    cerr << "Um inteiro foi passado: " << e << endl;
  }
  ... // Continua aqui depois de executar o catch.
}
```

Tratamento de exceções

- Abaixo mostramos o tratamento do erro de divisão por zero.

```
class ExcecaoDivZero {
  private:
    int nLinha;
  public:
    ExcecaoDivZero(int n);
    void msg();
};

ExcecaoDivZero::ExcecaoDivZero(int n) {
  nLinha = n;
}

void ExcecaoDivZero::msg() {
  cerr << "Tratador de erro foi chamado ..." << endl;
  cerr << "Erro na linha: " << nLinha << endl;
}
...
```

Tratamento de exceções

```
...
main() {
  int n, m;
  cout << "Numerador: ";
  cin >> n;
  cout << "Denominador: ";
  cin >> m;
  try { // inicio do bloco try
    if ( m==0 ) {
      // lança exceção, passando objeto ExcecaoDivZero
      throw ( ExcecaoDivZero(__LINE__) );
      //daqui para frente no try mais nada é executado
    }
    cout << "Divisão: "  << (float)n/m << endl;
  } // final do bloco try
  catch (ExcecaoDivZero& e) {
    e.msg();
  } }
```

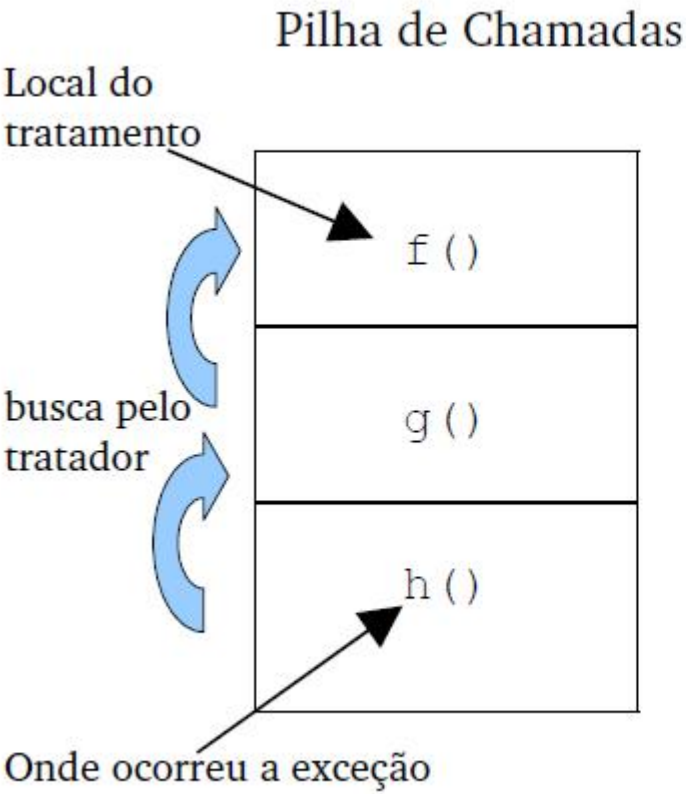
Tratamento de exceções

- O programador pode lançar qualquer objeto. Se a função onde a exceção foi lançada não possuir uma cláusula `catch` capaz de tratar a exceção, então é examinada a função que a chamou para verificar se esta possui um tratador. Esse busca continua nos níveis da pilha de chamadas até que seja encontrado um tratador.

- Se alguma exceção gerada não for capturada o programa será encerrado e uma mensagem padrão será exibida no dispositivo de saída.

Tratamento de exceções

- A figura abaixo mostra a sequência da busca por uma cláusula `catch` na pilha de chamadas.




```
void f(){
    try { g();}
    catch(exc1)
    {...}
}

void g()
{
    h(0);
    ...
}

void h(int p)
{
    if (p==0) throw exc1();
    ...
}
```

Tratamento de exceções

- Para capturar todas as exceções basta usar “...” no parâmetro da cláusula `catch`:

```
try {}
catch(...){}
```

- No entanto, tenha o cuidado de colocar a cláusula `catch(...)` após todas as outras cláusulas `catch`. Caso contrário, estas últimas nunca serão executadas, pois `catch(...)` captura todas as exceções. O exemplo a seguir mostra um trecho de código que captura qualquer exceção lançada.

```
int qualquer(int n, int d) {
  try {
    fazAlgumaCoisa();
  } catch(erro1) {
    cerr <<”ocorreu o erro 1!\n”;
  } catch(...) {
    cerr <<”ocorreu algum erro!\n”;
  }
}
```

Tratamento de exceções

- No tratamento da exceção (`catch`), o programador pode inserir código para relançar a exceção para um nível acima na cadeia de execução. Isto pode ser desejável se somente parte do tratamento puder ser realizado na função corrente, sendo necessário reenviar a exceção para níveis anteriores na pilha de chamada para finalizar o tratamento. O exemplo abaixo mostra um trecho de código que relança uma exceção.

```
int qualquer(int n, int d) {
  try {
    fazAlgumaCoisa();
  } catch(erro1) {
    cerr <<"ocorreu o erro 1!\n";
    throw; // relancamento da excecao
  }
}
```

Tratamento de exceções

➔ Exceções podem ser lançadas por componentes do C++. Neste caso, o sistema lança uma das exceções padrões da linguagem. São elas:

Exceção	Descrição
`bad_alloc`	lançado pelo `new` se alocação falhar
`bad_cast`	lançado por `dynamic_cast`
`bad_exception`	lançado quando um tipo de exceção diferente do esperado é lançado
`bad_typeid`	lançado por `typeid` (por ex. Na tentativa de obter classe através de ponteiro para `NULL`)
`ios_base::failure`	A classe failure define a classe base para os tipos de todos os objetos lançados como exceções, por funções na biblioteca de iostreams, para relatar erros detectados durante operações de buffer de fluxo.

Tratamento de exceções

- Exemplo:

```
try {
  int *myarray = new int[1000];
} catch (bad_alloc&) {
  cerr << "Erro ao alocar memória." << endl;
}
```

Tratamento de exceções

- Exemplo 2:

```
main() {
  double *ptr[ 100 ];
  try { // tentativa para alocar memória:
    // aloca memória para ptr[ i ]; new lança
    // bad_alloc em caso de falha
    for ( int i = 0; i < 100; i++ ) {
      ptr[ i ] = new double[ 5000000 ];
      cout << "Alocado 5000000 de doubles em ptr[ "
           << i << " ]\n";
    }
  } // fim try
  // trata bad_alloc exception:
  catch ( bad_alloc &me ) {
    cerr << "Ocorreu exceção: " << me.what() << endl;
  }
}
```

Tratamento de exceções

➔ Por fim, o tratamento de exceções deve ser utilizado como projetado, ou seja, como exceção e não como uma estrutura de controle do fluxo normal do programa.