Tratamento de Exceções

Prof. Valdir Pedrinho de Tomin Junior, Eng.

Universidade Federal de Santa Catarina
Centro Tecnológico de Joinville
EMB5631 – Programação III

e-mail: valdir.pedrinho@ufsc.br

23 de outubro de 2018

Introdução

Motivação

- Uma exceção é uma indicação de um problema que ocorre durante a execução de um programa;
- O tratamento de exceções permite aos programadores criar aplicativos que podem resolver (ou tratar) exceções;
- Em muitos casos, o tratamento de uma exceção permite que um programa continue executando como se nenhum problema tivesse sido encontrado;
- Um problema mais grave poderia impedir um programa de continuar executando normalmente, em vez de requerer que o programa notificasse o usuário sobre o problema antes de terminar de uma maneira controlada;
- O tratamento de exceções permite que o programador remova da 'linha principal' de execução do programa o código de tratamento de erro, o que aprimora a clareza do programa e fortalece sua modificabilidade.

Introdução

Quando usar Tratamento de Exceções

- O tratamento de exceções é um processo projetado para erros síncronos, que ocorrem quando uma instrução executa
  - Array fora do intervalo;
  - Overflow aritmético;
  - Divisão por zero;
  - Parâmetros de função inválidos;
  - Alocação de memória malsucedida.

Quando não usar Tratamento de Exceções

- O tratamento de exceções não é projetado para processar erros associados com os eventos assíncronos
  - Chegadas de mensagem de rede;
  - Cliques de mouse;
  - Pressionamentos de tecla.

Mecanismos para o Tratamento de Exceções

Uma Classe de Exceções

- A classe derivada da classe **exception** da *Standard Library* (definida no arquivo de cabeçalho <**exception**>) – é a classe básica padrão C++ para representar erros de tempo de execução;
- A classe **exception** é a classe básica padrão C++ para todas as exceções;
- Cada classe de exceções que deriva direta ou indiretamente de **exception** contém a função virtual **what**, que retorna uma mensagem de erro do objeto de exceção;
- Não é necessária derivar uma classe de exceções personalizada, mas isso permite a utilização da função virtual **what** para obter uma mensagem de erro apropriada;

Mecanismos para o Tratamento de Exceções

Palavra-chave **try**

- O C++ fornece blocos **try** para permitir o tratamento de exceções;
- O bloco **try** inclui instruções que poderiam causar exceções e instruções que devem ser ignoradas se ocorrer uma exceção;
- Se ocorrer uma exceção como o resultado de uma instrução em um bloco **try**, o bloco **try** expira e o programa procura o primeiro **handler catch** que pode processar o tipo de exceção que ocorreu;
- Se o bloco **try** completar sua execução com sucesso, então o programa ignora os **handler catch** e o controle do programa continua com a primeira instrução depois do último **catch** que se segue a esse bloco **try**.

Mecanismos para o Tratamento de Exceções

Handler catch

- As exceções são processadas por **handlers catch**, que capturam e tratam exceções;
- Cada **handler catch** inicia com a palavra-chave **catch** e especifica em parênteses um parâmetro de exceção que representa o tipo de exceção que o **handler catch** pode processar;
- O **handler catch** captura uma referência ao objeto criado por uma instrução **throw**;
- Em geral, um **handler catch** informa ao usuário sobre o erro, registra esse erro em um arquivo de log, termina o programa elegantemente ou tenta uma estratégia alternativa para realizar a tarefa que falhou.

Mecanismos para o Tratamento de Exceções

Palavra-chave **throw**

- A palavra-chave **throw** é usada para lançar a exceção;
- **throw** é seguida por um operando que representa o tipo de exceção a lançar;
- Um operando **throw** pode ser um objeto ou assumir o valor de um inteiro ou o valor de uma expressão;
- Obviamente, o comando de lançamento de exceção deve ser realizado antes da instrução que provocará o erro.

Mecanismos para o Tratamento de Exceções

Resumindo

- Dentro do bloco **try** são colocadas as instruções sensíveis a erros síncronos;
- O comando **throw** é usado para lançar uma exceção para este erro;
- O **handler catch** captura a exceção e o erro é tratado.

Exemplo

Exemplo 16.1-2 do *DEITEL. C++, Como programar. 5ª ed.*

Mecanismos para o Tratamento de Exceções

Relançando uma exceção

É possível que um **handler** de exceção, no recebimento de uma exceção, decida que não pode processar essa exceção ou que pode processá-la apenas parcialmente. Nesses casos, o **handler** de exceção pode adiar o tratamento de exceções (ou talvez uma parte dele) para outro **handler** de exceção. Em qualquer caso, o **handler** alcança isso relançando a exceção via a instrução **throw**.

Exemplo

Exemplo 16.3 do *DEITEL. C++, Como programar. 5ª ed.*

Mecanismos para o Tratamento de Exceções

Especificações de exceção

- Uma especificação de exceção opcional (também chamada lista **throw**) enumera uma lista de exceções que uma função pode lançar;
- Considere o seguinte exemplo:

Exemplo

```
int someFunction( double value )
  throw ( ExceptionA, ExceptionB, ExceptionC
   )
{// corpo da função}
```

A especificação de exceção indica que a função **someFunction** pode lançar exceções de tipos **ExceptionA**, **ExceptionB** e **ExceptionC**;

Mecanismos para o Tratamento de Exceções

Especificações de exceção

- Uma função pode lançar somente exceções dos tipos indicados pela especificação ou exceções de qualquer tipo derivado desses tipos;
- Se a função lança uma exceção que não pertence a um tipo especificado, a função **unexpected** é chamada;
- Colocar **throw()** – uma especificação de exceção vazia – depois da lista de parâmetros de uma função declara que a função não lança exceções. A tentativa de lançamento chamaria a função **unexpected**.

Processando exceções inesperadas

- A função **unexpected** chama a função registrada junto à função **set_unexpected**;
- A função **terminate** é chamada por padrão;

Mecanismos para o Tratamento de Exceções

Processando exceções inesperadas

- A função **terminate** é chamada quando:
  - O mecanismo de exceção não pode localizar um **catch** correspondente para uma exceção lançada;
  - Um destrutor tenta lançar uma exceção durante o desempilhamento;
  - Uma tentativa de relançar uma exceção é feita quando não há nenhuma exceção sendo atualmente tratada;
  - Uma chamada à função **unexpected** chama a função **terminate** por padrão.
- A função **set_terminate** pode especificar a função a ser invocada quando **terminate** for chamada;
- Por padrão, **terminate** chama **abort**, que termina o programa sem chamar os destrutores de quaisquer objetos, o que pode levar a vazamentos de recursos;
- A função **set_terminate** e a função **set_unexpected** retornam um ponteiro para a última função chamada por **terminate** e **unexpected**.

Mecanismos para o Tratamento de Exceções

Desempilhamento de pilha

- Quando uma exceção é lançada mas não capturada em um escopo particular, a pilha de chamadas de função é desempilhada e uma tentativa de capturar a exceção é feita no próximo bloco **try** externo;
- Desempilhar implica que todas as variáveis locais nessa função são destruídas e o controle retorna à instrução que originalmente invocou essa função;
- Se um bloco **try** incluir essa instrução, uma tentativa de capturar a exceção com **catch** é feita, caso contrário, novo desempilhamento;
- Se nenhum **handler catch** capturar essa exceção, a função **terminate** é chamada.

Exemplo

Exemplo 16.4 do *DEITEL. C++, Como programar. 5ª ed.*

Tratamento de Exceções na Alocação Dinâmica de Memória

Três formas de lidar com o **new**

- **new** retorna 0;
- **bad_alloc** é lançado;
- Função **set_new_handler**
  - Essa função aceita como seu argumento um ponteiro para uma função que não aceita argumentos e retorna **void**.

Exemplos

Exemplo 16.5, 16.6 e 16.7 do *DEITEL. C++, Como programar. 5ª ed.*

Classe **auto_ptr**

Classe **auto_ptr** e alocação de memória dinâmica

- Se ocorrer uma exceção depois de uma alocação de memória bem-sucedida, mas antes de a instrução **delete** executar, um vazamento de memória poderia ocorrer;
- O C++ fornece o *template* da classe **auto_ptr** no arquivo de cabeçalho <**memory**> para lidar com situações exceções com o gerenciamento de memória dinamicamente alocada;
- Um objeto da classe **auto_ptr** mantém um ponteiro para a memória dinamicamente alocada;
- Quando um destrutor de objeto **auto_ptr** é chamado esse realiza uma operação **delete** em seu membro de dados de ponteiro;
- O *template* da classe **auto_ptr** fornece os operadores sobrecarregados * e -> para que um objeto **auto_ptr** possa ser utilizado da mesma maneira que uma variável de ponteiro regular.

Classe **auto_ptr**, Exceções e polimorfismo

Exemplo de uso de **auto_ptr**

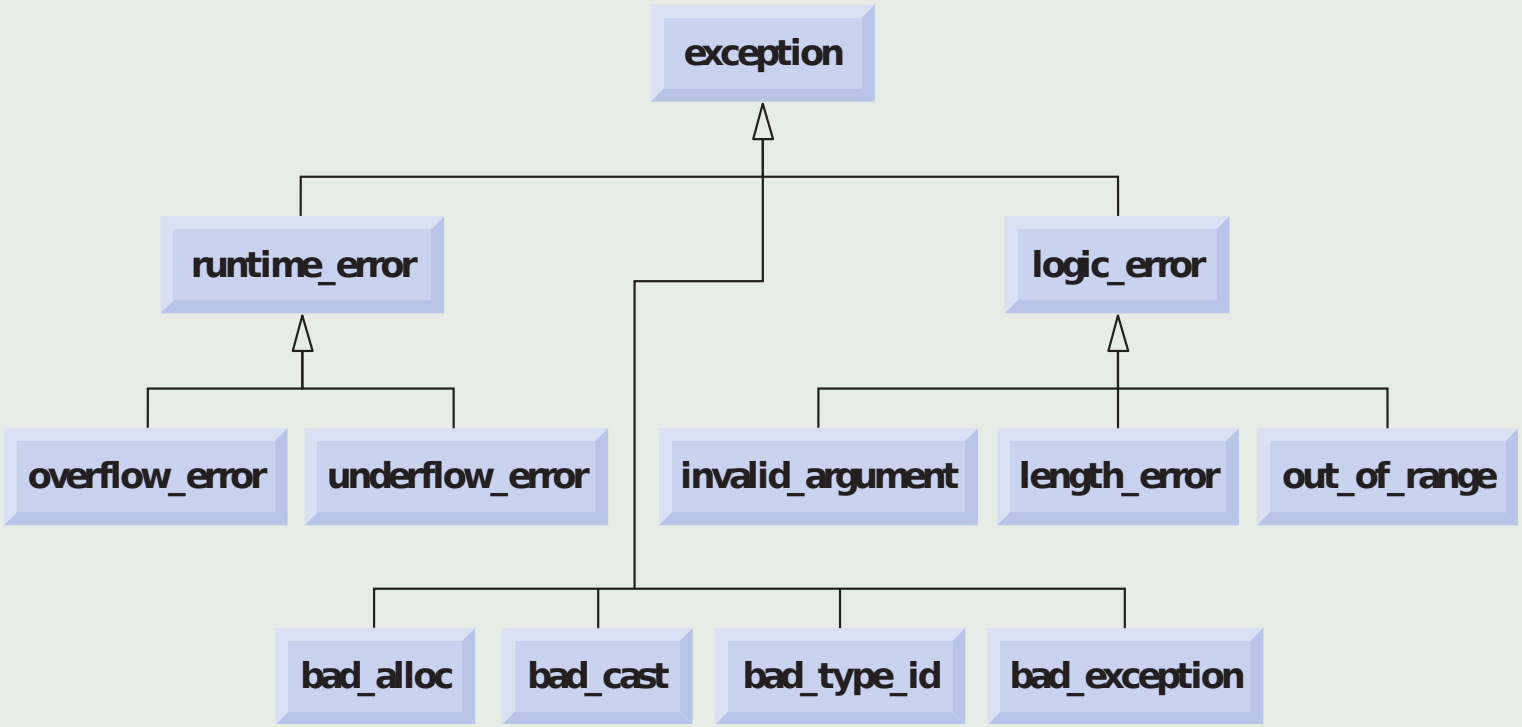
Exemplo 16.8-10 do *DEITEL. C++, Como programar. 5ª ed.*

Polimorfismo em classes de exceções

- Várias classes de exceções podem ser derivadas de uma classe básica comum;
- Se um **handler catch** captura um ponteiro ou referência para um objeto de exceção de um tipo de classe básica, ele também pode capturar um ponteiro ou referência para todos os objetos de classes publicamente derivadas dessa classe básica – isso permite processamento polimórfico de erros relacionados.

Exceções da biblioteca-padrão

Hierarquia de exceções



Lançamentos de Exceções

- **bad_alloc** é lançada por **new**;
- **bad_cast** é lançada por **dynamic_cast**;

Exceções da biblioteca-padrão

Lançamentos de Exceções

- **bad_typeid** é lançada por **typeid**;
- **bad_exception** é lançada quando uma exceção inesperada ocorre. A função **unexpected** pode lançar **bad_exception** em vez de terminar a execução do programa (por padrão);
- **logic_error** é a classe básica para exceções de erros de lógica
  - **invalid_argument** indica que um argumento inválido foi passado para uma função;
  - **length_error** indica que um comprimento maior que o tamanho máximo permitido para o objeto sendo manipulado;
  - **out_of_range** indica que um valor excedeu seu intervalo permitido de valores.
- **runtime_error** é a classe básica para erros de tempo de execução
  - **overflow_error** indica um resultado maior do que pode ser armazenado;
  - **underflow_error** indica um resultado menor do que pode ser armazenado.

Outras técnicas de tratamento de erro

Formas de lidar com erro, além de tratamento de exceções

- Algoritmo do avestruz (programa falha);
- Abortar o programa;
- Teste a condição de erro, emita uma mensagem de erro e chame **exit**

Referências e Exercícios

Livro Texto

Paul Deitel e Harvey Deitel.**C++: Como programar**. 5ªed. Ed. Prentice Hall Brasil. 2006.

Conteúdo dessa Aula

- Capítulo 16

Exercícios do Livro

Exercícios do capítulo 16.