Laboratório de programação II

Tratamento de exceções

Edson Moreno

edson.moreno@pucrs.br

http://www.inf.pucrs.br/~emoreno

Sumário

- Exceções
  - Introdução
  - Quando Utilizar Exceções?
  - Comandos para tratamento de exceção
  - Modelo de Terminação
  - Erros comuns
  - Classes de Exceções da Biblioteca Padrão

Introdução

- Uma **exceção** é uma indicação de um problema que ocorre durante a execução

- O **tratamento de exceções** permite que os programas sejam
  - Mais robustos
  - Tolerantes a falhas

- Os erros detectados durante a execução são processados
  - O programa trata o problema e pode continuar executando;
  - Ou o programa trata o problema e pode terminar sua execução

Exceções

- Em programas com muitas classes criadas por diferentes programadores, se ocorrer um erro de execução, qual classe o tratará?
  - Como estabelecer a comunicação entre os métodos das diferentes classes?

- Se uma função precisa enviar uma mensagem de erro, ela "**lança**" um objeto representando o erro para fora dela;

- Se a função que a chamou não "**capturar**" este objeto, ele será enviado para quem a chamou
  - E assim por diante, até que alguma função o **capture**.

Exemplo 1

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

int main(){
	int n=(int)90000000000, *ptr;

	ptr = new int [n];
	ptr[0] = 5;

	cout<<"Endereco: "<<(void*) ptr <<endl;

	delete [] ptr;

	return 0;
}
```

Exemplo 2

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

int main(){
        int n=(int)90000000000, *ptr;
        ptr = new int [n];
        if(ptr){
                ptr[0] = 5;
                cout<<"Endereco: "<<(void*) ptr <<endl;
                delete [] ptr;
        }
        else
                cout<<"Memoria insuficiente!" <<endl;
        return 0;
}
```

Exceções

- Nenhum destes dois programas evitam o erro causado quando a linha do operador ***new*** for executada
  - Compilação ok;
  - Não haverá tempo nem para executar o ***if*** que vem depois, no segundo código.

- O ideal seria "***tentar***" executar a instrução ***new***
  - Se der certo, ok;
  - Caso contrário, "***lançar***" um objeto com a exceção para que alguma função o "***capture***" e o processe.

Comandos para tratamento de exceção

- Estas ações podem ser realizadas com o uso de exceções;
- Temos 3 palavras chave:
  - *try* (*tentar*);
  - *throw* (*lançar*);
  - *catch* (*capturar*).

try

- A instrução ***try***
  - Define blocos de código em que exceções possam ocorrer
- O bloco de código
  - Precedido pela palavra try e é delimitado por { e };
  - As instruções que podem implicar em exceções e todas as instruções que não podem ser executadas em caso de exceção fazem parte do bloco de código.
  - É possível ter blocos ***try*** dentro de outro bloco ***try***.

```
try {
   funcao();
}
catch(exception &e) {
    // Código de tratamento da exceção
}
```

throw

- A instrução ***throw*** lança uma exceção
  - Indica que houve um erro;
  - É criado um objeto, que contém a informação sobre o erro
    - Logo, podemos criar uma classe que defina o erro
    - Ou utilizar uma existente.
  - Posteriormente, outro trecho de código capturará este objeto e tomará a atitude adequada.

```
int funcao(float numero) {
    // Codigo da função
    if (/* Condição de erro */)
        throw exception();
    //...
}
```

catch

- Exceções são tratadas por manipuladores *catch*
  - Capturam e processam as exceções.
- Pelo menos um *catch* deve seguir um bloco *try*
  - O bloco de código é precedido pela palavra ***catch*** seguido de parênteses e é delimitado por { e };
  - Dentro dos parênteses há os parâmetros da exceção
    - Representa o tipo de exceção que o *catch* pode processar.
  - O *catch* cujos parâmetros sejam de tipos iguais ao da exceção será executado;
  - O valor default para um catch é **...**
- Normalmente um *catch* imprime mensagens de erro, termina o programa elegantemente ou tenta refazer a operação que lançou a exceção.

try, throw e *catch*

- Um engano comum é achar que em todo tratamento de exceções é obrigatório usar pelo menos um *try*, um *throw* e um *catch* juntos
  - Recomendável é utilizar juntos o *try* e o *catch* para manipular as exceções;
  - O *throw* é utilizado somente para lançar exceções, não para manipulá-las.

Quando Utilizar Exceções?

- Erros síncronos (na execução de uma instrução), tais como:
  - Índice de vetor fora dos limites;
  - *Overflow* aritmético (valor fora dos limites do tipo);
  - Divisão por zero;
  - Parâmetros inválidos;
  - Alocação de memória.

- Exceções não devem ser utilizadas para erros assíncronos, tais como:
  - Erros de I/O;
  - Cliques de mouse e pressionamento de teclas.

Exemplo Try, throw, catch

```
#include <iostream>
using namespace std;
int fat(int val){
  if(val<0) throw “Argumento inválido”;
  if(val==0 || val==1) return 1;
  return val*fat(val-1);
}
int main(int argc, char** argv) {
        try{
                cout << "O fatorial de -5 é "<< fat(-1) << endl;
        }
        catch(const char* e){
          cerr << "Error(char): " << e << endl;
        }
        return 0;
}
```

Exemplo Try, throw, catch

```
#include <iostream>
using namespace std;
int fat(int val){
  if(val<0) throw -1;
  if(val==0 || val==1) return 1;
  return val*fat(val-1);
}
int main(int argc, char** argv) {
        try{
                cout << "O fatorial de -5 é "<< fat(-1) << endl;
        }
        catch(int e){
          cerr << "Error(int): " << e << endl;
        }
        return 0;
}
```

Exemplo Try, throw, catch

```
#include <iostream>
using namespace std;
int fat(int val){
  if(val<0) throw NULL;
  if(val==0 || val==1) return 1;
  return val*fat(val-1);
}
int main(int argc, char** argv) {
        try{
                cout << "O fatorial de -5 é "<< fat(-1) << endl;
        }
        catch(...){
          cerr << "Error - Tratador de exceções GENÉRICO" << endl;
        }
        return 0;
}
```

Exemplo Try, multi catch

```
int main(int argc, char** argv) {
        try{
                cout << "O fatorial de -5 é "<< fat(-1) << endl;
        }
        catch(int e){
          cerr << "Error(int): " << e << endl;
        }
        catch(const char *e){
          cerr << "Error(char*): " << e << endl;
        }
        catch(...){
          cerr << "Error - Tratador de exceções GENÉRICO" << endl;
        }
        return 0;
}
```

Erros Comuns

- Não pode haver código entre um bloco ***try*** e um ***catch***
  - É um erro de sintaxe.
- Cada ***catch*** só pode ter um único parâmetro
  - Uma lista de parâmetros é um erro de sintaxe.
- É um erro de lógica capturar o mesmo tipo de exceção em dois ***catch*** diferentes;
- Após o tratamento da exceção, achar que o fluxo de execução volta para o ponto em que a exceção foi lançada.

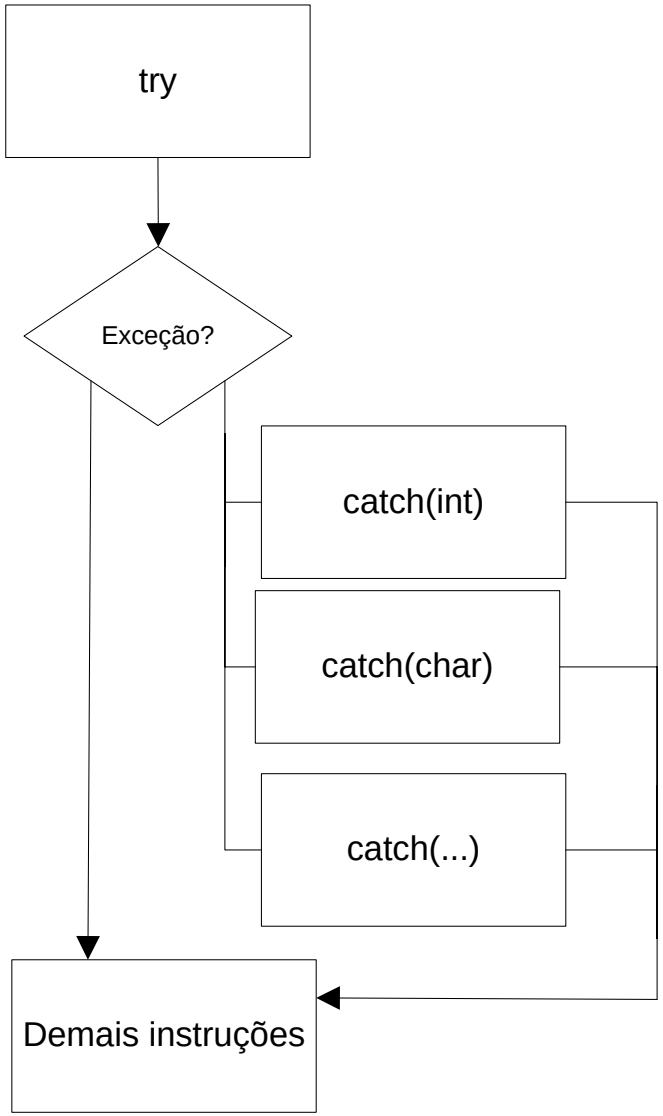
Modelo de Terminação

- Se em um bloco *try*, uma exceção ocorre:
  - O bloco termina imediatamente;
  - O primeiro *catch* cujo tipo seja igual (ou uma classe derivada) do tipo da exceção é executado;
  - Quando o *catch* termina, suas variáveis são destruídas e o fluxo de execução passa para a primeira instrução depois do último *catch* relacionado ao *try* que lançou a execução.
    - O que é chamado de **modelo de terminação**.
  - Há outro modelo em que o fluxo passa para o ponto logo após ao lançamento da exceção
    - **Modelo de resumo**.

Modelo de Terminação

- Se ocorrer uma exceção em um *try* e não houver *catch* correspondente, ou se não houver *try*
  - A função que contém a instrução com erro termina;
  - O programa tenta encontrar um bloco *try* aberto com a chamada da função;
  - Caso a exceção seja realmente inesperada, é executada uma chamada ***abort***
    - Não há nenhum bloco *try* que a contenha;
    - O programa termina sem chamar destrutores.
- Se não ocorrer uma exceção, o bloco *try* termina e todos os *catch* são ignorados.

Modelo de Terminação



Exceções padrão em C++

- Class exception
  - Método ***what***: mostra a mensagem de erro

```
class Excecao : public exception {
    const char * message;
    Excecao(){}
public:
    Excecao(const char * s) throw() : message(s){}
    const char * what() const throw() {return
message;}
};
```

Exceções padrão da stdlib	Descrição
bad_alloc	Enviada pelo new na falha de alocação
bad_cast	Enviada por dynamic_cast quando falha um cast dinâmico
bad_exception	Enviado por erros específicos de exceções dinâmicas
bad_typeid	Enviado por typeid
bad_function_call	Enviado por objetos com funções vazias
bad_weak_ptr	Enviado pelo shared_ptr quando passado um weak_ptr

Exceções padrão em C++

- Outras classes derivadas de exception
  - logic_error: informa erros lógicos do programa, os quais poderiam ser detectados antes do programa executar

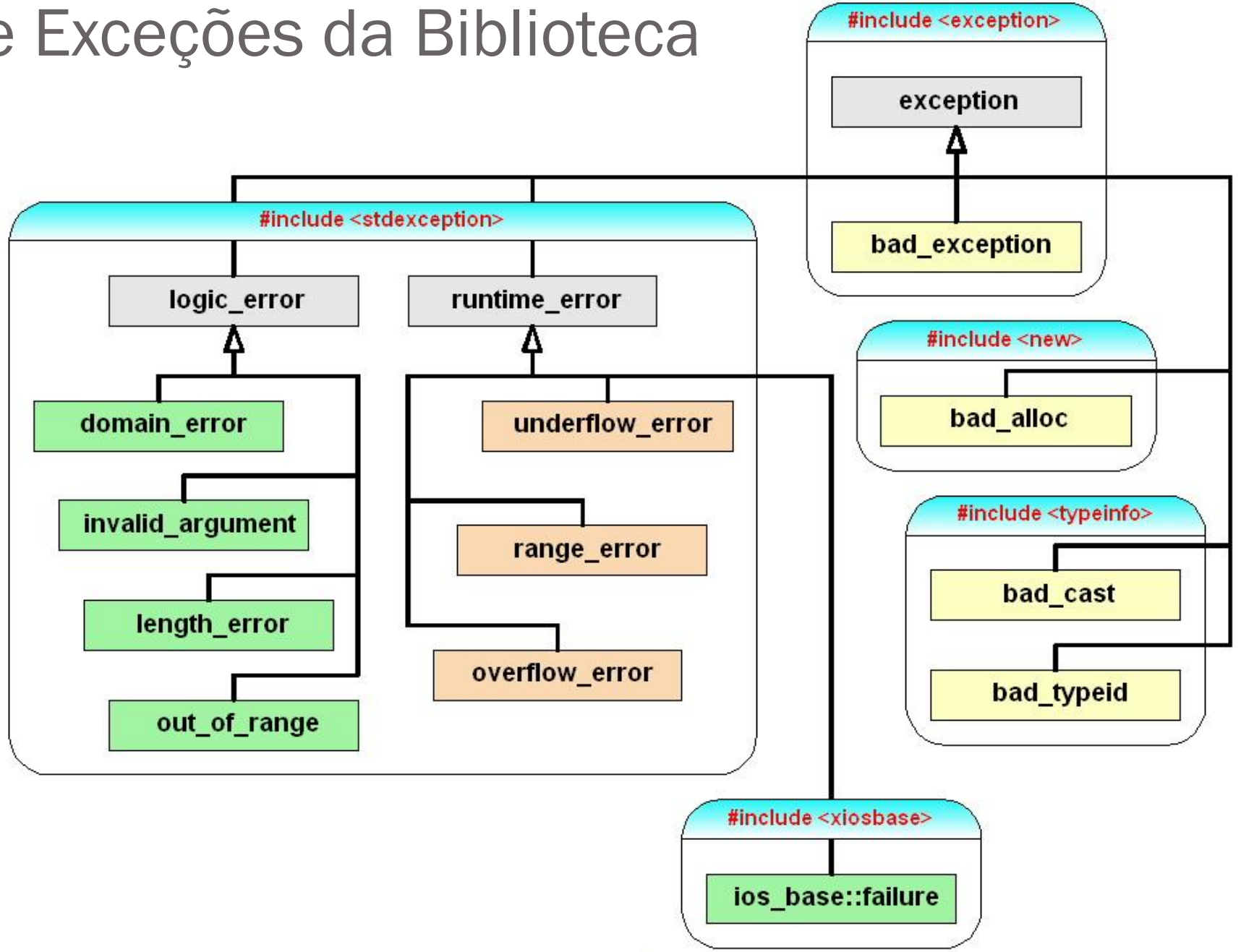
Exceções de logic_error	Descrição
Invalid_argument	Passagem de argumento inválido para a função que lançou a exceção
Length_error	Tentativa de produzir um objeto cujo tamanho é > que o valor máximo aceito para o tipo
Out_of_range	Reporta a existência de um valor fora da faixa permitida
Bad_cast	Lançada a partir de uma falha de cast dinâmico
Bad_typeid	Relata um ponteiro p nulo em uma expressão typeid(*)

Exceções padrão em C++

- Outras classes derivadas de exception
  - runtime_error: informa erros de runtime do programa, os quais só poderiam ter sido detectados durante exceção

Exceções de run_error	Descrição
Range_error	Relata violação de uma faixa de valores
Overflow_error	Relata um overflow aritmético
Underflow_error	Relata um underflow aritmético

Classes de Exceções da Biblioteca Padrão



Exceções padronizadas

```
#include <iostream>
#include <exception>
using namepace std;


int main(){
  double *ptr;
  try{
    while (1){
      cout << "Tentando Alocar...\n";
      ptr = new double[500000];
    }
  }
  catch (bad_alloc E) {
    cout << "Faltou Memoria...\n" << endl;
  }


  for(int i=0; i< 100;i++)
    cout << "Fim..." << endl;

  return 0;
}
```

Extenção de exceções padronizadas

```
// using standard exceptions
#include <iostream>
#include <exception>
using namespace std;
class myexception: public exception{
  virtual const char* what() const throw()  {
    return "My exception happened";
  }
} myex;
int main () {
  try{
    throw myex;
  }
  catch (exception& e){
    cout << e.what() << '\n';
  }
  return 0;
}
```

Exercícios

- Crie um programa com uma função que realiza a divisão entre dois número. Trate a situação na qual o divisor possa assumir o valor zero
- Crie uma situação semelhante ao exemplo em que ocorra um erro por estouro (overflow). Utilize variáveis que estouram mais comumente, como short int ou byte