Exceções em C++

Roland Teodorowitsch

Programação Orientada a Objetos - ECo - Curso de Engenharia de Computação - PUCRS

8 de novembro de 2023

Exceções

Exceções em C++

- Nem sempre é possível criar métodos ou funções que retornam valores que possam ser identificados como códigos de erros
- Exceções fornecem uma forma de reagir a circunstâncias excepcionais nos programas (tais como erros de execução), associando ações à ocorrência de cada circunstância
- Em C++:
  - uma parte de código é colocada sob inspeção (bloco `try`)
  - se algo de errado ocorrer na execução deste código, uma exceção será lançada de forma explícita (comando `throw`) ou implícita
  - se exceções forem lançadas, elas podem ser capturadas em blocos `catch`

Exemplo 1

```
#include <iostream>

using namespace std;

int fatorial(int n) {
    if (n < 0) throw "fatorial de número negativo!";
    if (n == 0 || n == 1) return 1;
    return n * fatorial(n-1);
}

int divisao(int a, int b) {
    if (b == 0) throw "divisão por zero!";
    return a / b;
}

int main() {
    try {
        cout << fatorial(4) << endl;
        cout << divisao (4,0) << endl;
        cout << fatorial(-1) << endl;
        cout << fatorial(0) << endl;
    }
    catch (const char *e) {
        cerr << "\nERRO: " << e << endl;
    }
    return 0;
}
```

Funcionamento

- Todo o código para o qual se deseja tratamento deve estar em um bloco `try`
- `throw` aceita um parâmetro (no exemplo anterior, cadeias de caracteres), que é passado como argumento para os blocos de tratamento
- Os blocos de tratamento são declarados com o comando `catch` (logo após o bloco `try`) e aceitam um único parâmetro
  - o tipo do parâmetro de um `catch` é importante e deve corresponder ao tipo do parâmetro que tiver sido lançado
  - um `catch` só captura um `throw` se ambos forem do mesmo tipo
  - é possível ter vários blocos de tratamento, um após o outro
- Se forem utilizadas reticências (`...`) como parâmetro do `catch`, então este tratador captura qualquer exceção lançada (usa-se esta construção como um tratador padrão para capturar todas as exceções não capturadas por outros tratadores)

Exemplo 2

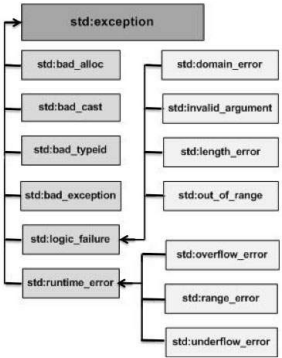
```
#include <iostream>

using namespace std;

int main () {
    try {
        cout << "ANTES" << endl;
        throw 20;
        throw "ERRO␣DESCONHECIDO";
        cout << "DEPOIS" << endl;
    }
    catch (int e)  {
        cerr << "Ocorreu␣uma␣exceção␣de␣número␣" << e << "." << endl;
    }
    catch (...) {
        cerr << "Erro␣desconhecido..." << endl;
    }
    return 0;
}
```

Exceções Padronizadas de C++

- C++ provê uma lista de exceções padronizadas definidas em <excpetion> e que podem ser utilizadas nos programas
- Estas exceções estão organizadas na seguinte hierarquia de classes



Fonte: https://www.tutorialspoint.com/cplusplus/images/cpp_exceptions.jpg

Exceções Personalizadas

- É possível definir exceções personalizadas derivando a classe `exception` e eventualmente sobrescrevendo seus métodos
- O próximo exemplo mostra como usar a classe `exception` para implementar uma exceção personalizada

Exemplo 3

```
#include <iostream>
#include <exception>

using namespace std;

class minhaExcecao : public exception {
    virtual const char* what() const throw() {
        return "Minha exceção aconteceu!";
    }
};

int main () {
    try {
        cout << "ANTES" << endl;
        throw minhaExcecao();
        cout << "DEPOIS" << endl;
    }
    catch (exception &e) {
        cerr << e.what() << endl;
    }
    return 0;
}
```

Exercício

Exercício 1

1. Nas páginas a seguir você encontra a implementação de um Tipo Abstrato de Dados (TAD) para uma pilha de inteiros (arquivos `IntStack.hpp`, `IntStack.cpp` e `IntStackMain.cpp`). Esta classe (`IntStack`), possui os seguintes métodos em sua interface:
   - `IntStack(int mxSz = 10)`: construtor que recebe o tamanho máximo da pilha
   - `~IntStack()`: destrutor
   - `int size()`: retorna o número de elementos da pilha
   - `int maxSize()`: retorna o número máximo de elementos suportado pela pilha
   - `bool isEmpty()`: retorna `true`, se a pilha estiver vazia, ou `false`, em caso contrário
   - `bool isFull()`: retorna `true`, se a pilha estiver cheia, ou `false`, em caso contrário
   - `void clear()`: esvazia a pilha
   - `bool push(const int e)`: insere o elemento no topo da pilha (retorna `true`, em caso de sucesso, ou `false`, se NÃO houver espaço)
   - `bool pop(int &e)`: remove e retorna (por referência) o elemento do topo da pilha (retorna `true`, em caso de sucesso, ou `false`, a pilha estiver vazia)
   - `bool top(&e)`: retorna (por referência) o elemento do topo da pilha, mas não o remove da pilha (retorna `true`, em caso de sucesso, ou `false`, a pilha estiver vazia)

   Modifique esta classe para que ela utilize `vector` da STL, *templates* (de forma que ela NÃO fique restrita a inteiros) e exceções para os casos em que alguma operação inválida seja executada sobre a pilha. Com isso a interface dos seguintes métodos será alterada:
   - `void push(T &)`: insere o elemento no topo da pilha
   - `T pop()`: remove e retorna o elemento do topo da pilha
   - `T top() const`: retorna o elemento do topo da pilha, mas não o remove da pilha

Exemplo de Implementação: IntStack.hpp

```
#ifndef _INTSTACK_HPP
#define _INTSTACK_HPP

#include <string>

using namespace std;

class IntStack {
private:
  int numElements;
  int maxElements;
  int *stack;
public:
  IntStack(int mxSz = 10);
  ~IntStack();
  int size() const;
  int maxSize() const;
  bool isEmpty() const;
  bool isFull() const;
  void clear();
  bool push(const int e);
  bool pop(int &e);
  bool top(int &e) const;
  string str() const; // APENAS PARA DEPURACAO
};
#endif
```

Exemplo de Implementação: IntStack.cpp

```
#include <sstream>
#include "IntStack.hpp"

IntStack::IntStack(int mxSz) {
  numElements = 0;   maxElements =  ( mxSz < 1 ) ? 10 : mxSz;
  stack = new int[maxElements];
}

IntStack::~IntStack() {  delete[] stack;  }
int IntStack::size() const { return numElements; }
int IntStack::maxSize() const { return maxElements; }
bool IntStack::isEmpty() const { return numElements == 0; }
bool IntStack::isFull() const { return numElements == maxElements; }
void IntStack::clear() { numElements = 0; }

bool IntStack::push(const int e) {
  if ( numElements == maxElements ) return false;
  else { stack[ numElements++ ] = e;  return true; }
}

bool IntStack::pop(int &e) {
  if ( numElements == 0 ) return false;
  else { e = stack[ --numElements ];  return true; }
}

bool IntStack::top(int &e) const {
  if ( numElements < 1 ) return false;
  else { e = stack[ numElements-1 ];  return true; }
}

string IntStack::str() const {
  int i;   stringstream ss;
  ss << "|";
  for (i=0; i<numElements; ++i) ss << stack[i] << "|";
  for (; i<maxElements; ++i) ss << "␣|";
  return ss.str();
}
```

Exemplo de Implementação: IntStackMain.cpp

```
#include <iostream>
#include "IntStack.hpp"

using namespace std;

void print(IntStack &stack) {
  cout << "␣␣" << stack.str() << "␣␣size=" << stack.size() << "/" << stack.maxSize() << "␣␣top=";
  int t;   bool res = stack.top(t);
  if (res) cout << t; else cout << "X";
  cout << "␣␣isEmpty=" << stack.isEmpty() << "␣␣isFull=" << stack.isFull() << endl;
}

int main() {
  int e;
  bool res;
  cout << "IntStack(4):␣";  IntStack stack(4);  print(stack);
  e = 1;  cout << "push(" << e << "):␣"; res = stack.push(e);  cout << (res?"OK␣␣":"ERRO");  print(stack);
  e = 2;  cout << "push(" << e << "):␣"; res = stack.push(e);  cout << (res?"OK␣␣":"ERRO");  print(stack);
  e = 3;  cout << "push(" << e << "):␣"; res = stack.push(e);  cout << (res?"OK␣␣":"ERRO");  print(stack);
  res = stack.pop(e); cout << "pop(" << e << "):␣␣";  cout << (res?"OK␣␣":"ERRO");  print(stack);
  e = 4;  cout << "push(" << e << "):␣"; res = stack.push(e);  cout << (res?"OK␣␣":"ERRO"); print(stack);
  e = 5;  cout << "push(" << e << "):␣"; res = stack.push(e);  cout << (res?"OK␣␣":"ERRO"); print(stack);
  e = 6;  cout << "push(" << e << "):␣"; res = stack.push(e);  cout << (res?"OK␣␣":"ERRO"); print(stack);
  res = stack.pop(e); cout << "pop(" << e << "):␣␣";  cout << (res?"OK␣␣":"ERRO");  print(stack);
  res = stack.pop(e); cout << "pop(" << e << "):␣␣";  cout << (res?"OK␣␣":"ERRO");  print(stack);
  res = stack.pop(e); cout << "pop(" << e << "):␣␣";  cout << (res?"OK␣␣":"ERRO");  print(stack);
  res = stack.pop(e); cout << "pop(" << e << "):␣␣";  cout << (res?"OK␣␣":"ERRO");  print(stack);
  res = stack.pop(e); cout << "pop(X):␣␣";  cout << (res?"OK␣␣":"ERRO");  print(stack);
  e = 7;  cout << "push(" << e << "):␣"; res = stack.push(e);  cout << (res?"OK␣␣":"ERRO"); print(stack);
  cout << "clear():␣OK␣␣";  stack.clear();  print(stack);
  return 0;
}
```