Introdução a Técnicas de Programação

Testando as funções

Prof. André Campos
DIMAp/UFRN

Como saber se um código funciona como esperado?

Podemos **provar formalmente** (matematicamente) que nosso algoritmo funciona (vocês verão rapidamente em outras disciplinas)

… ou podemos …

TESTAR

Ou seja, imaginar situações (entrada de dados) que o código possa “quebrar” e gerar expectativas para as suas respostas (saída de dados)

Testes de unidade

Organiza os testes sobre “unidades de código” (elementos independentes, o que permite testar de forma isolada)

- **funções**
- classes (métodos de classe)
- componentes

Metodologia de desenvolvimento - TDD (*Test Driven Development*)

Desenvolvimento dirigido a testes

Desenvolvimento dirigido a testes

1. Escreva casos de teste para o que você quer implementar
2. Crie uma versão simples o suficiente apenas para poder compilar
3. Compile, execute os testes e veja o que vai falhar
4. Implemente/altere a lógica para passar nos testes que falharam
5. Volte ao passo 3 enquanto houver teste que falhou
6. Se acha que pode melhorar:
    6.1. Melhore o código onde você identificou que pode melhorar
    6.2. Insira novos testes para tratar situações ainda não exploradas
    6.3. Volte ao passo 3

Testes de função usando a main()

Como fizemos até o momento, até funciona... mas não é o ideal

```
int maior(int a, int b, int c) {
  if (a > b and a > c) return a;
  if (b > c) return b;
  return c;
}

int main() {
  cout << maior(5, 2, 7) << endl;
  cout << maior(1, 8, 4) << endl;
  cout << maior(-7, -4, -1) << endl;
  return 0;
}
```

Chamada	Valor de retorno
`maior(5, 2, 7)`	7
`maior(1, 8, 4)`	8
`maior(-7, -4, -1)`	-1

Quais os problemas?!?!?

Ferramentas de auxílio a testes

Há várias! Vamos apresentar uma só: **doctest**

https://github.com/doctest/doctest

Uso do doctest

1. Baixe a última versão na pasta de trabalho

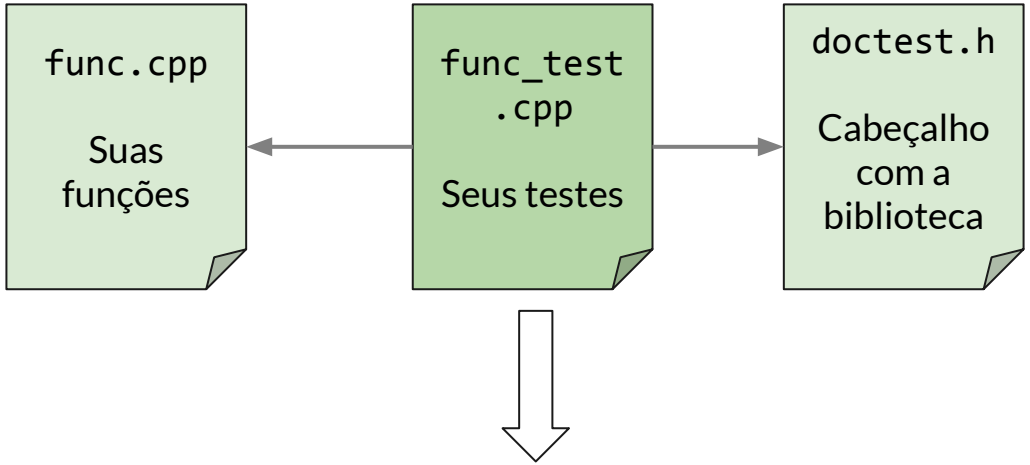
```
$ wget https://github.com/doctest/doctest/releases/download/v2.4.12/doctest.h
```

2. Crie dois arquivos:
   2.1. um arquivo com as funções que você vai desenvolver. Ex: `func.cpp`
   2.2. um arquivo com testes das funções. Ex: `func_test.cpp`

3. inclua o arquivo cabeçalho do doctest e o arquivo de suas funções no arquivo de testes

4. Use as “macro” definidas no doctest para testar suas funções

Uso do doctest




```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"
#include "func.cpp"

...
```

Configura o doctest para criar a função main()

Insere a biblioteca de testes doctest

Insere o arquivo com suas funções

Macros básicos do doctest

TEST_CASE(“Texto explicando o que está sendo testado”) { ... }
CHECK(expressão lógica do teste) → se verdade, então passou no teste

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"
#include "utils.cpp"

TEST_CASE("Testando a função que calcula o maior entre 3 valores") {
  CHECK(maior(5, 7, 1) == 7);
  CHECK(maior(-5, 0, -1) == 0);
  CHECK(maior(-5, -7, -1) == -1);
  CHECK(maior(8, 1, 1) == 8);
  CHECK(maior(0, 0, 0) == 0);
}
```

Exemplo (com sucesso)

func.cpp

```
int maior(int a, int b, int c) {
  if (a > b and a > c) return a;
  if (b > c) return b;
  return c;
}
```

func_test.cpp

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"
#include "func.cpp"

...
```

```
$ g++ func_test.cpp
$ ./a.out
[doctest] doctest version is "2.4.11"
[doctest] run with "--help" for options
===============================================
[doctest] test cases: 1 | 1 passed | 0 failed | 0 skipped
[doctest] assertions: 5 | 5 passed | 0 failed |
[doctest] Status: SUCCESS!
```

Exemplo (sem sucesso)

func.cpp

```
int maior(int a, int b, int c) {
  if (a > b and a > c) return a;
  if (b > c) return b;
  return  b ;
}
```

func_test.cpp

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"
#include "func.cpp"

...
```

```
$ g++ func_test.cpp
$ ./a.out
...
func_test.cpp:5:
TEST CASE:  Testando a função que calcula o maior entre 3 valores
func_test.cpp:8: ERROR: CHECK( maior(-5, -7, -1) == -1 ) is NOT correct!
  values: CHECK( -7 == -1 )
===============================================================================
```

Comparação de números reais

Comparação de números de tipos ponto flutuante

3.100000001 é igual a **3.1 ???**

Usar uma aproximação, por exemplo de 2 casas decimais

3.100000001 é igual a **3.10** !!!

```
CHECK(3.1 == doctest::Approx(3.100000001));
```

...ou, se necessário,...

```
CHECK(3.1 == doctest::Approx(3.1001).epsilon(.01));
```

Práticas

Escreva testes e solução para uma função para classificar o tipo de um triângulo de acordo com os seus lados. A função deve receber 3 valores reais (double), que correspondem às dimensões de seus lados, e deve retornar um valor inteiro, com um desses 4 possíveis valores: 0 se for um triângulo equilátero (3 lados iguais), 1 se for um triângulo isósceles (2 lados iguais), 2 se for um triângulo escaleno (3 lados diferentes) ou 3 se não formar um triângulo.

Entrada	Saída
`10.5  10.5  10.5`	`0`
`3.2  3.2  4.1`	`1`
`5.0  3.2  3.2`	`1`
`3.2  3.1  3.2`	`1`

Entrada	Saída
`3.2  4.9  4.1`	`2`
`5.0  3.1  2.0`	`2`
`5.0  3.1  2.0`	`2`
`4.7  4.9  13.0`	`3`

Entrada	Saída
`-1.0 3.2 5.4`	`3`
`5.0  3.1  0`	`3`
`8.1  3.1  3.1`	`3`
`1.1  4.5  2.7`	`3`

Práticas

Escreva testes e solução para uma função que recebe uma string e retorna a versão “CamelCase” dela.

Entrada	**Saída**
De tudo ao meu amor serei atento	DeTudoAoMeuAmorSereiAtento
Antes e com tal zelo e sempre e tanto	AntesEComTalZeloESempreETanto
Que mesmo em face do maior encanto	QueMesmoEmFaceDoMaiorEncanto
Dele se encante mais meu pensamento	DeleSeEncanteMaisMeuPensamento

Obs: CamelCase é o formato de escrita na qual as palavras compostas ou frases (com espaços) são reescritas de forma que cada palavra é iniciada com maiúsculas e unidas sem espaços. Ex: “agora é a hora” → AgoraÉAHora

Práticas

Altere os arquivos da atividade anterior para incluir testes e solução para uma função que recebe uma string no formato “CamelCase” e retorne a versão “normal” dela.

Entrada	**Saída**
DeTudoAoMeuAmorSereiAtento	De tudo ao meu amor serei atento
AntesEComTalZeloESempreETanto	Antes e com tal zelo e sempre e tanto
QueMesmoEmFaceDoMaiorEncanto	Que mesmo em face do maior encanto
DeleSeEncanteMaisMeuPensamento	Dele se encante mais meu pensamento

Obs: CamelCase é o formato de escrita na qual as palavras compostas ou frases (com espaços) são reescritas de forma que cada palavra é iniciada com maiúsculas e unidas sem espaços. Ex: “agora é a hora” → AgoraÉAHora

Práticas

Escreva testes e o código de uma função para testar se **a rainha** (em um jogo de Xadrez) que se encontra em uma determinada posição do tabuleiro pode se mover para outra determinada posição.

Por exemplo, se a rainha se encontra na posição **c2**, ela pode se mover para a posição **g6** (é um movimento válido), mas não pode mover para **g7** (não é um movimento válido).

Obs: a rainha se movimenta em qualquer direção (horizontal, vertical e diagonais) e qualquer número de casas

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	**c2**	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

Práticas

A imagem ao lado ilustra (em verde) os movimentos válidos para uma rainha que se encontra na posição c2.

Entrada	Saída
c2 g6	verd.
c2 g7	falso
b5 h5	verd.
b5 a3	falso
a1 h8	verd.

Entrada	Saída
d7 g4	verd.
d7 e5	falso
f2 d2	verd.
f2 d1	falso
e8 e2	verd.

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	**c2**	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

Práticas

Escreva testes e o código de uma função para testar se **um cavalo** (em um jogo de Xadrez) que se encontra em uma determinada posição do tabuleiro pode se mover para outra determinada posição.

Por exemplo, se o cavalo se encontra na posição **c2**, ele pode se mover para a posição **d4** (é um movimento válido), mas não pode mover para **c4** (não é um movimento válido).

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	**c2**	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

Obs: o cavalo se movimenta em no formato de “L” usando 3 saltos: 2 pra “frente” (horizontal ou vertical) e 1 pro “lado”.

Práticas

A imagem ao lado ilustra (em verde) os movimentos válidos para um cavalo que se encontra na posição c2.

Entrada	Saída
c2 b4	verd.
c2 a4	falso
b5 d4	verd.
b5 c4	falso
a1 b3	verd.

Entrada	Saída
d7 e5	verd.
d7 e4	falso
f2 d1	verd.
f2 d2	falso
e8 d6	verd.

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	**c2**	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1