IMD0029 – ESTRUTURAS DE DADOS BÁSICAS I

PROF. EIJI ADACHI M. BARBOSA

Roteiro de Implementação – Breve Introdução a Testes Automatizados

No decorrer da disciplina, usaremos extensivamente Testes de Software Automatizados. Não se preocupem, pode parecer complexo, mas a responsabilidade de implementar os testes será minha; vocês precisam apenas compreender como usar os testes que eu irei disponibilizar. O objetivo desta tarefa é, portanto, fazer uma breve introdução a como usaremos testes de software automatizados como uma forma de auxiliar a implementação dos algoritmos e estruturas de dados vistos ao longo da disciplina.

De modo bastante simplificado, Teste de Software é uma forma sistemática de executarmos o programa que estamos implementando para verificarmos que ele está correto. Dizemos que um programa está correto se ele atende a sua especificação, isto é, se ele produz os resultados esperados. Testes de Software Automatizados, também chamados de Testes Executáveis, são programas de computador que verificam se determinados módulos de software estão corretos.

Junto a este roteiro, estão disponíveis os seguintes arquivos fonte: main.cpp, Math.cpp e Math.hpp. Os arquivos Math.hpp e Math.cpp definem o módulo “Math”, um módulo bastante simples que provê funções que realizam as operações aritméticas básicas entre números inteiros. Releve a simplicidade deste módulo, pois o objetivo desta atividade é analisarmos o arquivo main.cpp, o qual implementa os testes executáveis que irão verificar a corretude do módulo Math.

Abra o arquivo main.cpp e analise seu código. Você observará que há uma função main que invoca duas outras funções: testAdd e testSubtract. As funções testAdd e testSubtract implementam os testes executáveis das funções add e subtract providas pelo módulo Math. Inspecione com cuidado o código das funções testAdd e testSubtract, pois cada uma implementa testes executáveis de uma forma diferente.

Na função testAdd, o teste executável é todo implementado “na mão”. Perceba que a estrutura da função é bastante repetitiva: nós invocamos a função add do módulo Math, salvamos o resultado retornado numa variável chamada result e verificamos num bloco if se o valor da variável result é o valor esperado para uma operação de soma. Por exemplo, no início da função testAdd nós fazemos a invocação add(1, 1) e, em seguida, testamos no bloco if se o valor retornado é igual a 2, que é o valor esperado para uma soma entre 1 e 1.

Agora analise o código da função testSubtract. Perceba que a função testSubtract utiliza a função assert da biblioteca padrão do C. A função assert é utilizada para nos auxiliar na implementação de testes executáveis, diminuindo a repetição de código, como visto na função testAdd. A função assert funciona da seguinte forma: ela

recebe como argumento uma expressão booleana. Se esta expressão for verdadeira, a função assert retorna normalmente e o programa continua sua execução; se esta expressão for falsa, a função assert interrompe a execução do programa e exibe uma mensagem de erro na tela. Observe na função testSubtract como nós utilizamos a função assert para realizar um test executável. Observe que a primeira linha da função testSubtract é assert(subtract(1, 1) == 0). Nós podemos ler esta linha da seguinte forma: assegure-se que o valor retornado pela chamada subtract(1, 1) é igual a 0. Ou seja, nós queremos verificar que ao subtrairmos 1 de 1, o valor retornado seja igual a 0. Caso a chamada subtract(1, 1) de fato retorne 0, então a expressão subtract(1, 1) == 0 será verdadeira e, portanto, a função assert irá retornar normalmente, dando continuidade à execução. Caso a chamada subtract(1, 1) não retorne 0, então a expressão subtract(1, 1) == 0 será falsa e, portanto, a função assert irá interromper a execução do programa. Neste caso, ao observamos uma interrupção na execução do teste executável, nós saberemos que a implementação da função subtract não está conforme o esperado e, portanto, nós identificamos uma falha no nosso módulo sob teste.

Para melhor compreender o funcionamento da função assert, compile o código do main.cpp e do módulo Math para gerar um executável. Para isto, digite no terminal o seguinte comando:

```
g++ Math.cpp main.cpp -o testMath -Wall -pedantic -std=c++11
```

O comando acima invoca o compilador g++ para que sejam compilados e linkados os arquivos Math.cpp e main.cpp, gerando o arquivo executável nomeado testMath. As opções –Wall –pedantic e –std=c++11 são configurações opcionais passados ao compilador, mas que deverão sempre ser utilizados ao longo da disciplina. Repito: estas configurações deverão ser usadas sempre. Faça uma breve pesquisa para compreender para que servem as opções –Wall –pedantic e –std=c++11.

Uma vez gerado o arquivo executável, vá até o terminal e execute-o com o seguinte comando:

```
./testMath
```

Após executar o arquivo, você deverá ver no terminal as seguintes mensagens:

```
All add-tests passed!
Assertion failed: (subtract(1, 1) == 0), function testSubtract, file main.cpp,
line 65.
```

A primeira linha no terminal foi impressa pela função testAdd. Ela indica que todos os testes sobre a função add foram realizados com sucesso e, portanto, pode-se assumir que não se encontraram defeitos nesta função. Já a segunda linha impressa no terminal foi impressa pela função assert. A segunda linha deve ser interpretada da seguinte forma: a verificação de que a chamada subtract(1, 1) deveria ser igual a 0 falhou; esta verificação está sendo realizada na função testSubtract do arquivo main.cpp na linha 65. Em outras palavras, nós identificamos um defeito na função subtract do módulo Math. Para corrigi-lo, basta modificar no arquivo Math.cpp o operador aritmético da função subtract de * para –. Depois de fazer esta correção, compile o código, e executando os testes

outra vez. Desta vez, você deverá ver as seguintes mensagens no terminal:

```
All add-tests passed!
All subtract-tests passed!
All tests passed!
```

Parabéns! Você completou a primeira parte deste mini-tutorial sobre testes de software automatizados. Ao longo da disciplina, vocês receberão outros testes similares aos implementados nas funções testAdd e testSubtract.

Agora, faça os seguintes exercícios complementares para praticar um pouco o que acabamos de ver:

1) Refaça a função testAdd de modo que os testes sejam implementados usando a função assert.

2) Crie as funções multiply e divide no módulo Math, as quais recebem dois int e retornam a multiplicação e divisão destes números, respectivamente. Você também deverá implementar as respectivas funções de teste (testMultiply e testDivide) no arquivo main.cpp. Para implementar as funções de teste, utilize a função assert, similar ao que é implementado na função testSubtract.

3) Crie uma função factorial no módulo Math, a qual recebe um número inteiro e retorna o seu fatorial. Esta função deverá ser iterativa; não deve ser recursiva. Você também deverá implementar a função de test testFactorial no arquivo main.cpp, também utilizando a função assert.

4) Modifique a função add para que ela realize a soma dos dois números de modo recursivo. Garanta que sua implementação recursiva passa pelos mesmos testes que a função passava anteriormente. Dica: somar X + Y é o mesmo que fazer X vezes a soma de Y + 1.

5) Modifique a função multiply para que ela realize a soma dos dois números de modo recursivo. Garanta que sua implementação recursiva passa pelos mesmos testes que a função passava anteriormente. Dica: multiplicar X * Y é o mesmo que fazer X vezes a soma de Y + Y.

6) Modifique a função factorial para que ela realize o cálculo do fatorial de modo recursivo. Garanta que sua implementação recursiva passa pelos mesmos testes que a função passava anteriormente.