Maykon dos Santos Gonçalves RA: 821653

Testes de Unidade (Unit Tests)

Para esse trabalho, decidi fazer um programa em C++, bem simples, que realiza conversões de unidade, sejam elas de temperatura ou de medidas. Para realizar os testes, utilizei o doctest, que é bem simples e rápido de usar, precisando apenas de um header file. E como framework de build, utilizei o Meson.

O programa em si consiste de 3 classes:

- Temperatura
- Medida
- Conversor

A main executa um simples menu para fazer conversões, e pode ser testado rodando `./programa` .

Os testes foram feitos de maneira bem simples, verificando se os métodos das classes estão funcionando corretamente através do Doctest. Foram elaborados 3 arquivos de teste:

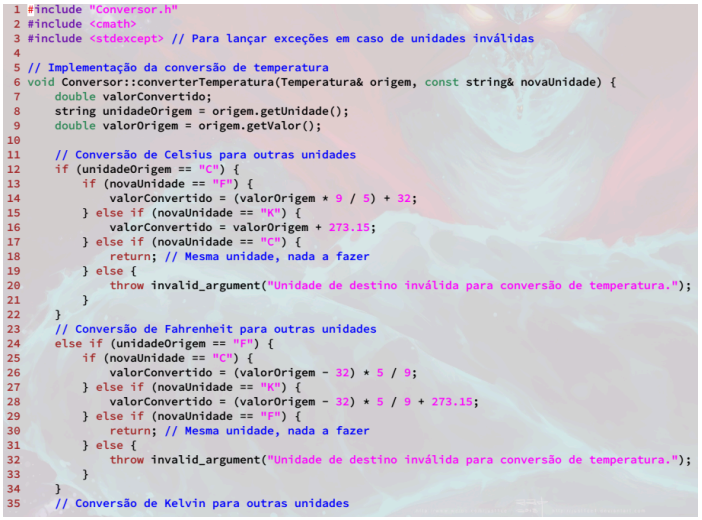
- test_temperatura.cpp
- test_medida.cpp
- test_conversor.cpp

A test_main.cpp é apenas para trazer os header files necessários para a execução dos testes, uma vez que o doctest exige uma main dedicada só pra isso. Isso também permite que o Meson execute individualmente cada teste.

O Programa

Tanto a classe Temperatura quanto a Medida são muito similares, contendo um atributo para valor e outro para unidade. Agora a classe Conversor, tem apenas dois métodos, um para converter Temperaturas, e outro para converter Medidas. As conversões são feitas no objeto original, uma escolha feita por simplicidade, uma vez que não há necessidade de guardar o valor anterior, evitando a criação de mais um objeto.

```cpp
#ifndef TEMPERATURA_H
#define TEMPERATURA_H

#include <string> // E: 'string' file not found
using namespace std;

class Temperatura {
private:
    double valor;       // Armazena o valor da temperatura
    string unidade;     // Armazena a unidade da temperatura (ex: "C", "F", "

public:
    // Construtor
    Temperatura(double valor = 0.0, string unidade = "C");

    // Métodos get
    double getValor() const;       // Retorna o valor da temperatura
    string getUnidade() const;     // Retorna a unidade da temperatura

    // Métodos set
    void setValor(double valor);   // Define o valor da temperatura
    void setUnidade(string unidade); // Define a unidade da temperatura

    // Método para imprimir os detalhes da temperatura
    void imprime() const;
};
```



```cpp
#include "Conversor.h"
#include <cmath>
#include <stdexcept> // Para lançar exceções em caso de unidades inválidas

// Implementação da conversão de temperatura
void Conversor::converterTemperatura(Temperatura& origem, const string& novaUnidade) {
    double valorConvertido;
    string unidadeOrigem = origem.getUnidade();
    double valorOrigem = origem.getValor();

    // Conversão de Celsius para outras unidades
    if (unidadeOrigem == "C") {
        if (novaUnidade == "F") {
            valorConvertido = (valorOrigem * 9 / 5) + 32;
        } else if (novaUnidade == "K") {
            valorConvertido = valorOrigem + 273.15;
        } else if (novaUnidade == "C") {
            return; // Mesma unidade, nada a fazer
        } else {
            throw invalid_argument("Unidade de destino inválida para conversão de temperatura.");
        }
    }
    // Conversão de Fahrenheit para outras unidades
    else if (unidadeOrigem == "F") {
        if (novaUnidade == "C") {
            valorConvertido = (valorOrigem - 32) * 5 / 9;
        } else if (novaUnidade == "K") {
            valorConvertido = (valorOrigem - 32) * 5 / 9 + 273.15;
        } else if (novaUnidade == "F") {
            return; // Mesma unidade, nada a fazer
        } else {
            throw invalid_argument("Unidade de destino inválida para conversão de temperatura.");
        }
    }
    // Conversão de Kelvin para outras unidades
```

O DocTest

O Doctest é um framework de tests para C++, baseado no Catch2, bem rápido, bem leve, com a vantagem de precisar de apenas um Header File para ser executado. A forma de construir testes nele é bem intuitiva, podendo adicionar TestCases e Subcases para aprofundar os testes.

Python

```cpp
#include "doctest.h"
#include "Medida.h"
using namespace std;

TEST_CASE("Test Medida Constructor and Getters") {
    Medida m(10.0, "metros");
    CHECK(m.getValor() == 10.0);
    CHECK(m.getUnidade() == "metros");
}

TEST_CASE("Test Medida Setters") {
    Medida m(0.0, "metros");
    m.setValor(32.8084);
    m.setUnidade("pés");
    CHECK(m.getValor() == 32.8084);
    CHECK(m.getUnidade() == "pés");
}
```

O Framework Meson

Após alguma pesquisa, escolhi o Meson por parecer ser simples, e uma boa alternativa ao Cmake, que é o mais comumente utilizado. A configuração dele é dada por apenas um arquivo .build

Python

```
project('conversor-app', 'cpp',
 default_options: ['cpp_std=c++17','b_coverage=true' ])
fs = import('fs') #To format Strings
# Include directories
doctest_inc = include_directories('include')
src_inc = include_directories('src')

# Source files
src_files = files(
 'src/Temperatura.cpp',
 'src/Medida.cpp',
 'src/Conversor.cpp',
)
src_main = files(
 'src/Main.cpp'
)

# Test files
test_files = files(
 'tests/test_main.cpp',  # Main test file with
DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
 'tests/test_temperatura.cpp',
 'tests/test_medida.cpp',
 'tests/test_conversor.cpp'
)

# Build the test executable
test_exe = executable(
 'conversor-tests',  # Single executable for all tests
 test_files + src_files,  # Include all test and source files
 include_directories: [doctest_inc, src_inc],  # Include directories
 dependencies: [], # No external dependencies needed for doctest
 cpp_args: ['--coverage'], #Enables gcov code coverage (doesn't seem to
work)
 link_args: ['--coverage']
)

# Build the program executable
prog_exe = executable(
 'programa',
 src_files + src_main
 )
```

```
# Register individual test cases with Meson
test('Temperatura Tests', test_exe, args: ['--test-case=Temperatura'])
#Those tak
e names from the test.cpp files
test('Medida Tests', test_exe, args: ['--test-case=Medida'])
test('Conversor Tests', test_exe, args: ['--test-case=Conversor'])
```

Muito prático, dá pra separar os testes das src files. No meu caso, como os testes dependem da definição das classes, tive que compilá-las juntamente aos testes. Surpreendente isso não causou loop nem compilações desnecessárias. Infelizmente, não consegui usar o Gcov para verificar o quanto do código os testes estavam cobrindo.

Ao dar build, o Meson já executa os testes, e informa se eles foram bem sucedidos ou não. Para uma visualização mais detalhada dos testes, é possível executar ‘conversor-tests’ dentro da pasta builddir.

Para instruções de como executar o programa e os testes, é só clonar o repositório e seguir o README:

https://github.com/sgmaykon/Unit-Tests---ES1