Tutorial Completo e Corrigido:
Testes Unitários em C++ com Catch2
Abordagem Prática para Windows com MSYS2

Documento de Referência Atualizado

15 de dezembro de 2025

Conteúdo

1 Introdução e Escopo

Este tutorial ensina como configurar um ambiente de testes unitários em C++ usando o framework Catch2, com foco em usuários Windows que utilizam Visual Studio Code como editor e MSYS2 como ambiente de compilação.

Informação

Abordagem deste tutorial:

- **Visual Studio Code:** Editor de código (escrever e salvar)
- **MSYS2 MINGW64:** Compilação e execução de testes

Esta é a abordagem mais confiável e sem problemas no Windows.

Atenção

Por que não executar no terminal integrado do VSCode?
O terminal integrado do VSCode no Windows tem limitações conhecidas com programas que usam formatação especial de saída (cores, posicionamento). Frameworks de teste como Catch2 frequentemente não exibem resultados corretamente neste ambiente. Por isso, usaremos o MSYS2 diretamente para compilação e execução.

2 Instalação e Configuração

2.1 Instalação do Visual Studio Code

1. Baixe e instale o Visual Studio Code de `https://code.visualstudio.com/`
2. Instale a extensão **C/C++ Extension Pack** da Microsoft

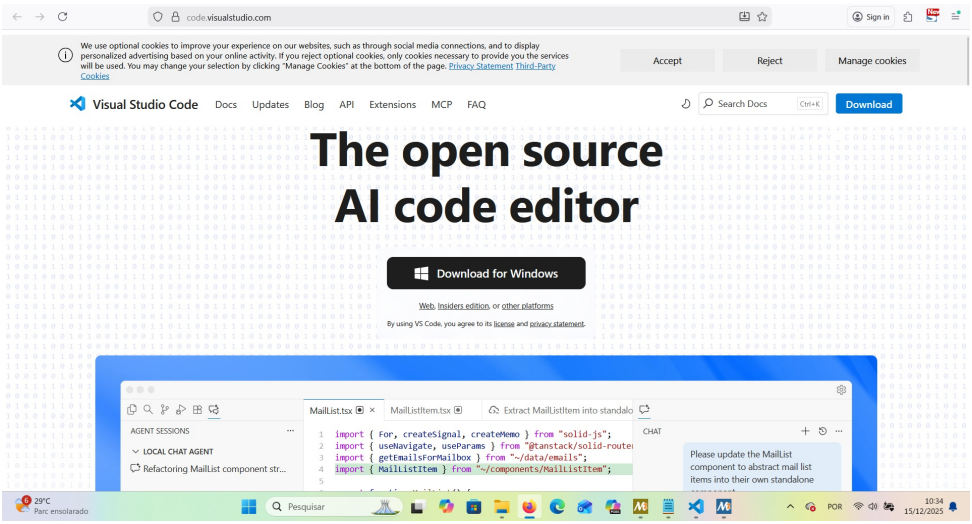

Figura 1: Download do Visual Studio Code.

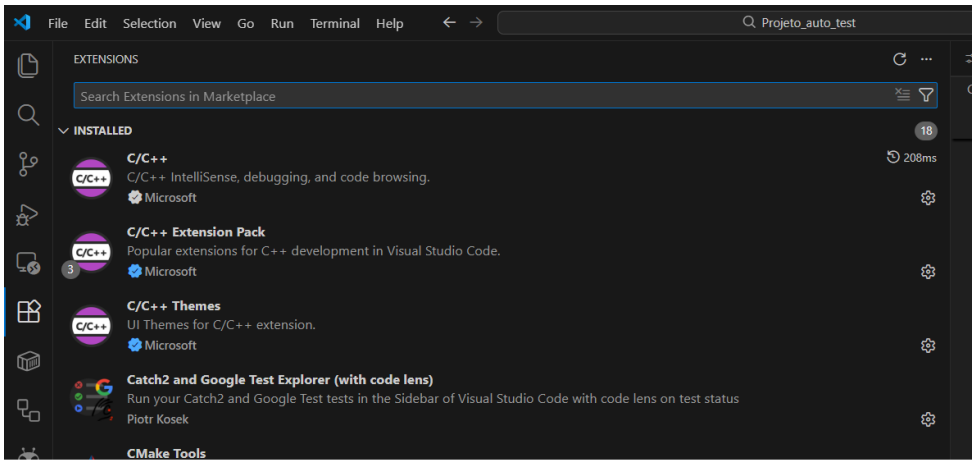

Figura 2: Extensões do VSCode.

2.2 Instalação do MSYS2

1. Baixe o instalador do MSYS2 em `https://www.msys2.org/`
2. Execute o instalador e siga as instruções (pasta padrão: `C:\msys64`)
3. Após a instalação, abra o terminal **MSYS2 MINGW64** ou **MSYS2 UCRT64** (ambos funcionam e a explicação é válida). Não use o MSYS2 básico!

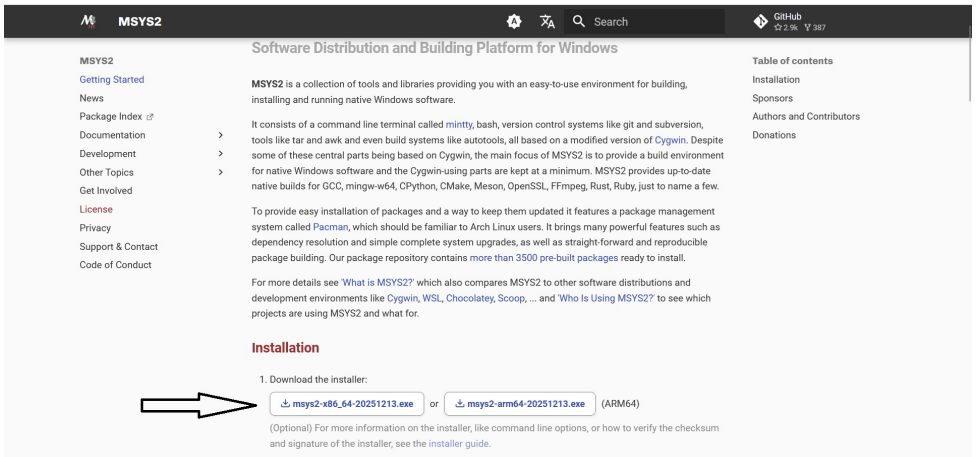

Figura 3: Terminal MSYS2 MINGW64.

Informação

Como encontrar o MSYS2 MINGW64:

- Pressione a tecla Windows
- Digite: `MINGW64`
- Clique em **MSYS2 MINGW64** (ícone roxo/azul)

2.3 Instalação do Compilador G++

No terminal **MSYS2 MINGW64**, execute:

Comando para Copiar

```
pacman -S mingw-w64-x86_64-gcc
```

Quando perguntado, digite `Y` e pressione Enter.

2.3.1 Verificação da Instalação

Após a instalação, verifique:

Comando para Copiar

```
g++ -version
```

Sucesso

Se aparecer a versão do compilador (ex: `g++ (Rev...) 15.2.0`), a instalação foi bem-sucedida!

Erro Comum

Erro: "bash: g++: command not found"
Isso significa que você está no terminal errado. Certifique-se de estar usando **MSYS2 MINGW64**, não o MSYS2 básico. Feche o terminal e abra o correto.

3 Estrutura do Projeto

3.1 Organização de Pastas

Crie a seguinte estrutura de diretórios:

```
PROJETO_AUTO_TEST/
|-- include/
|   |-- calculator.h
|   |-- catch.hpp
|-- src/
|   |-- Calculator.cpp   <<Codigo de producao
```

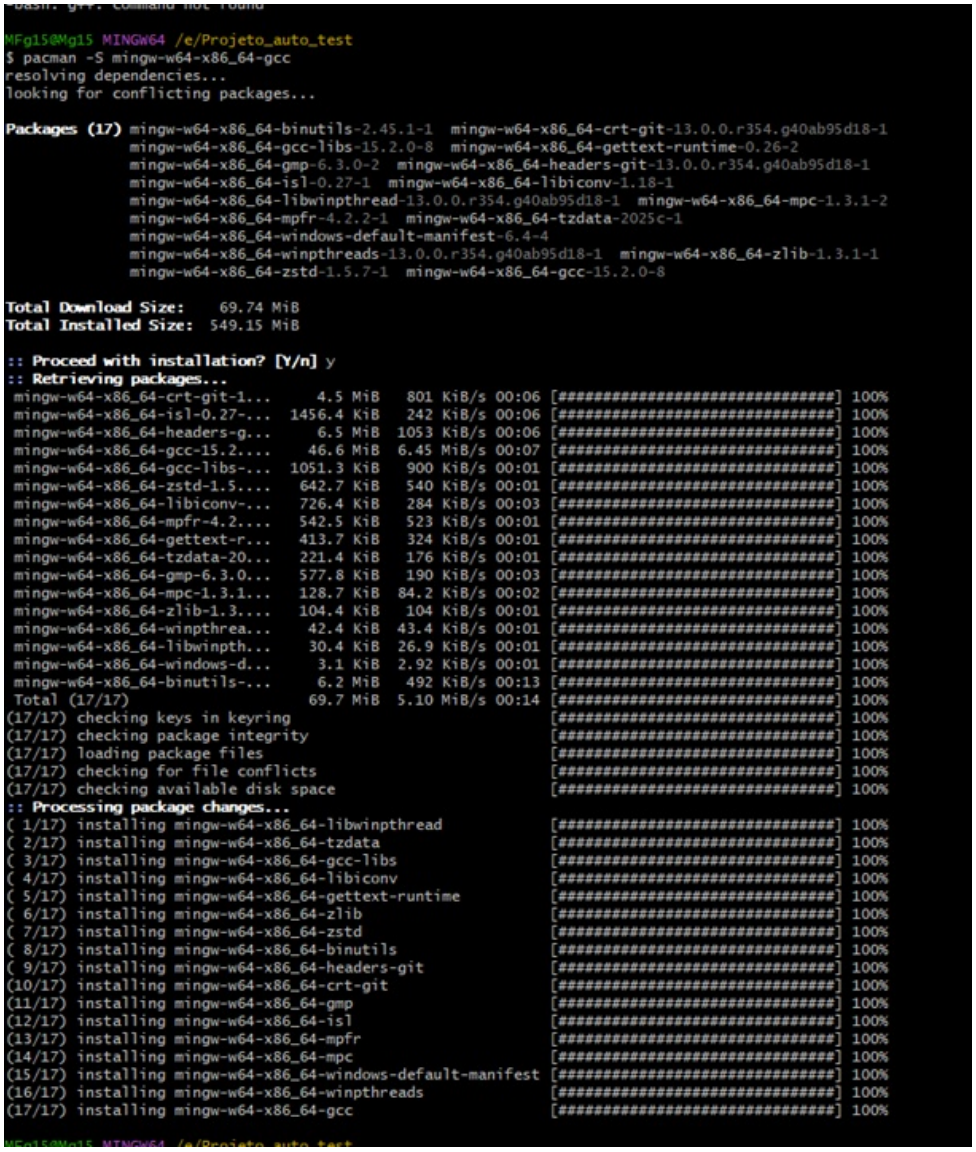

Figura 4: Instalação via pacman.

```
|-- tests/
    |-- test_calculator.cpp
    |-- calculator.cpp <<stub de teste
```

Listing 1: Estrutura do projeto

Informação

Significado de cada pasta:

- `include/`: Headers (interfaces) e bibliotecas externas
- `src/`: Código de produção (implementação)
- `tests/`: Código de testes unitários

3.2 Download do Catch2

1. Acesse: `https://github.com/catchorg/Catch2/tree/v2.x?tab=readme-ov-file`
2. Baixe a versão **v2.13.10** (single header), desça até achar
3. Procure por "`the latter version of the single header`" e lá você vai baixar o arquivo `catch.hpp` nos assets
4. Salve o arquivo na pasta `include/`

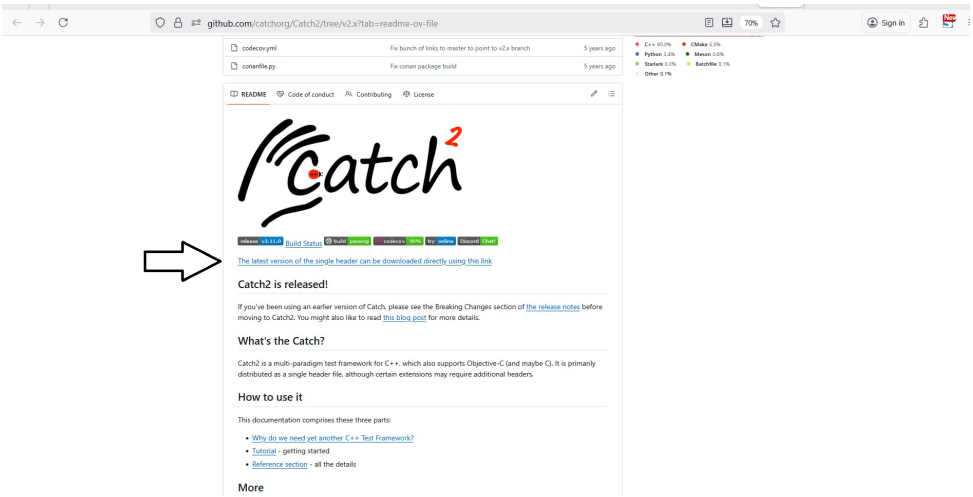

Figura 5: Download do Catch2 single header.

Atenção

Use a versão **v2.x** do Catch2 (não a v3.x). A v2 é single-header e mais simples para iniciantes.

4 Código-Fonte

4.1 Header: calculator.h

Crie o arquivo `include/calculator.h`:

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

class Calculator {
public:
    Calculator();
    double addition(double a, double b);
    double subtraction(double a, double b);
    double multiplication(double a, double b);

};

#endif // CALCULATOR_H
```

Listing 2: include/calculator.h

4.2 Implementação: calculator.cpp

Crie o arquivo `src/calculator.cpp`:

```
#include "../include/calculator.h"
#include <stdexcept>

Calculator::Calculator() {}

double Calculator::addition(double a, double b) {
    return a + b;
}

double Calculator::subtraction(double a, double b) {
    return a - b;
}

double Calculator::multiplication(double a, double b) {
    return a * b;
}
```

Listing 3: src/calculator.cpp

4.3 Testes: test_calculator.cpp

Explicação rápida: Cada teste feito nos Test Cases possui valores de entrada como no exemplo de adição são 2.0 e 3.0 ou -1.0 e 1.0 e valores esperados na saída colocados com == como 5.0 e 0.0, são com esses valores que a suite de testes vai verificar o código alvo. No caso da adição ele soma 2.0 e 3.0 e espera 5.0. Os outros testes fazem o mesmo com suas respectivas operações e valores de entrada e saída.

Crie o arquivo `tests/test_calculator.cpp`:

```
#define CATCH_CONFIG_MAIN
#include "../include/catch.hpp"
#include "../include/calculator.h"

TEST_CASE("Operacoes da calculadora", "[calculator]") {
    Calculator calc;

    SECTION("Testes de adicao") {
        REQUIRE(calc.addition(2.0, 3.0) == 5.0);
        REQUIRE(calc.addition(-1.0, 1.0) == 0.0);
        REQUIRE(calc.addition(0.0, 0.0) == 0.0);
    }

    SECTION("Testes de subtracao") {
        REQUIRE(calc.subtraction(5.0, 3.0) == 2.0);
        REQUIRE(calc.subtraction(0.0, 5.0) == -5.0);
    }

    SECTION("Testes de multiplicacao") {
        REQUIRE(calc.multiplication(4.0, 3.0) == 12.0);
        REQUIRE(calc.multiplication(-2.0, 3.0) == -6.0);
    }
}
```

Listing 4: tests/test_calculator.cpp

Informação

Entendendo o código de teste:

- `#define CATCH_CONFIG_MAIN`: Carrega a função main do catch automaticamente
- `TEST_CASE`: Define um conjunto de testes
- `SECTION`: Agrupa testes relacionados
- `REQUIRE`: Verifica se uma condição é verdadeira
- `[tags]`: Permitem filtrar testes na execução

5 Fluxo de Trabalho (Build e Test)

5.1 Workflow Completo

1. **Editar código:** Use o Visual Studio Code
2. **Salvar:** Ctrl + S no VSCode
3. **Alternar para MSYS2:** Alt + Tab ou clique na janela

4. **Compilar e testar:** Execute comandos no MSYS2
5. **Ver resultados:** Analise a saída no MSYS2
6. **Voltar ao VSCode:** Alt + Tab para fazer ajustes

5.2 Navegação no MSYS2

Abra o **MSYS2 MINGW64** e navegue até seu projeto:

Comando para Copiar

```
cd /e/Projeto_auto_test
```

Para verificar se está no lugar certo:

Comando para Copiar

```
pwd
```

Para listar arquivos e pastas:

Comando para Copiar

```
ls
```

Atenção

Atenção à sintaxe de caminhos:

- Windows: `E:\Projeto_auto_test`
- MSYS2: `/e/Projeto_auto_test`

Use barras normais (/) e letras minúsculas para drives.

5.3 Compilação

Execute o comando de compilação:

Comando para Copiar

```
g++ -std=c++11 -o test_calculator tests/*.cpp -I./include
```

Informação

Nota: O asterisco (*) é um curinga que inclui todos os arquivos .cpp da pasta tests.

Informação

Explicação dos parâmetros:

- `-std=c++11`: Usa o padrão C++11
- `-o test_calculator`: Nome do executável gerado
- `tests/*.cpp`: Todos os arquivos .cpp da pasta tests (inclui calculator.cpp e test_calculator.cpp)
- `-I./include`: Pasta onde estão os headers

Sucesso

Compilação bem-sucedida:
Se não aparecer nenhum erro, o arquivo `test_calculator.exe` foi criado na pasta raiz do projeto.

Erro Comum

Erros comuns de compilação:

- `No such file or directory`: Verifique os caminhos dos arquivos
- `undefined reference`: Esqueceu de incluir algum .cpp
- `catch.hpp not found`: Verifique se o arquivo está em include/

5.4 Execução dos Testes

Após compilar com sucesso, execute os testes:

Comando para Copiar

```
./test_calculator -success -durations yes
```

5.4.1 Saída Esperada

```
1 ===============================================
2 All tests passed (6 assertions in 3 test cases)
3 ===============================================
```

Listing 5: Exemplo de saída dos testes

5.5 Comando Combinado (Build + Test)

Para compilar e testar em um único comando:

Comando para Copiar

```
g++ -std=c++11 -o test_calculator tests/*.cpp -I./include &&
./test_calculator -success -durations yes
```

Informação

O operador `&&` significa: "se a compilação for bem-sucedida, então execute os testes". Se houver erro de compilação, os testes não serão executados.

6 Comandos Avançados do Catch2

6.1 Opções de Visualização

Mostrar todos os testes (incluindo sucessos):

Comando para Copiar

```
./test_calculator -success
```

Mostrar duração de cada teste:

Comando para Copiar

```
./test_calculator -success -durations yes
```

Modo verboso:

Comando para Copiar

```
./test_calculator -s
```

6.2 Filtragem de Testes

Executar apenas testes com tag específica:

Comando para Copiar

```
./test_calculator [calculator]
```

6.3 Geração de Relatórios

6.3.1 Relatório em XML (JUnit)

Comando para Copiar

```
./test_calculator -r junit -o relatorio.xml
```

Este comando gera um arquivo XML compatível com ferramentas de CI/CD.

7 Dicas e Boas Práticas

7.1 Organização de Testes

- Use `TEST_CASE` para agrupar funcionalidades relacionadas
- Use `SECTION` para subdividir casos de teste
- Use tags descritivas: `[componente][funcionalidade]`
- Mantenha testes independentes (não compartilhem estado)

7.2 Nomenclatura

- Nomes de testes devem ser descritivos
- Use português ou inglês consistentemente
- Exemplo: `TEST_CASE("Calculadora: Operação Basica")`

8 Troubleshooting

8.1 Problemas Comuns

Erro Comum

Erro: "g++: command not found"
Solução: Você está no terminal errado. Use **MSYS2 MINGW64**, não o MSYS2 básico.

Erro Comum

Erro: "No such file or directory"
Solução:

- Verifique se está na pasta raiz do projeto (`pwd`)
- Verifique se os caminhos no comando estão corretos
- Use `ls` para listar arquivos e confirmar estrutura

Erro Comum

Erro: "undefined reference to Calculator::..."
Solução: Você esqueceu de incluir o arquivo `calculator.cpp` no comando de compilação.

Erro Comum

Erro: "catch.hpp: No such file or directory"
Solução:

- Verifique se `catch.hpp` está na pasta `include/`
- Verifique o parâmetro `-I./include` no comando

9 Conclusão

Este tutorial apresentou a abordagem mais confiável para desenvolvimento com testes unitários em C++ no Windows:

- **VSCode:** Editor poderoso para escrever código
- **MSYS2 MINGW64:** Ambiente robusto para compilação e execução
- **Catch2:** Framework simples e eficaz para testes unitários

Esta separação de responsabilidades elimina problemas de compatibilidade e garante que os testes sempre exibam resultados corretamente.

9.1 Recursos Adicionais

- Documentação oficial do Catch2: `https://github.com/catchorg/Catch2`
- Tutorial do MSYS2: `https://www.msys2.org/docs/what-is-msys2/`
- Boas práticas de testes: `https://github.com/isocpp/CppCoreGuidelines`