Tutorial de Testes Unitários em C com µTest

Abordagem para Windows

Arthur Oliveira

15 de dezembro de 2025

Conteúdo

1 Introdução e Escopo

Este tutorial ensina como configurar e utilizar um ambiente de testes unitários em C utilizando o framework **µTest** (`utest.h`), com foco em usuários Windows que utilizam o **Visual Studio Code** como editor e o **MSYS2** (MINGW64 ou UCRT64) como ambiente de compilação e execução.

O µTest é um framework de testes unitários leve, portátil e baseado apenas em C, ideal para aplicações embarcadas, firmware, bibliotecas C e projetos que não utilizam C++.

Informação

Abordagem adotada neste tutorial:

- **Visual Studio Code:** Editor de código-fonte (edição, navegação e análise estática)
- **MSYS2 MINGW64 ou UCRT64:** Compilação e execução dos testes
- **Linguagem:** C (C99 ou superior)
- **Framework de testes:** µTest (`utest.h`)

Esta abordagem é simples e estável, de facil implementação.

Atenção

Sobre o terminal integrado do VS Code
Embora o terminal integrado do VS Code funcione para compilação e execução simples, este tutorial recomenda o uso direto do terminal **MSYS2 MINGW64 ou UCRT64** para evitar problemas de ambiente, variáveis de caminho incorretas e diferenças entre shells.
Usar o MSYS2 diretamente garante que o compilador (`gcc`), o linker e as bibliotecas estejam corretamente configurados.

1.1 Instalação do Visual Studio Code

1. Baixe e instale o Visual Studio Code em `https://code.visualstudio.com/`

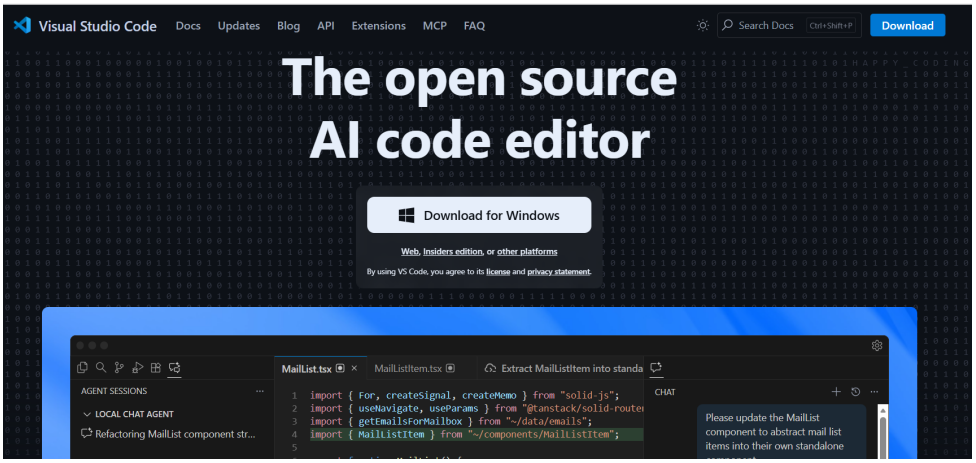

Figura 1: Página de download do Visual Studio Code.

2. Instale as extensões da Microsoft listadas abaixo.

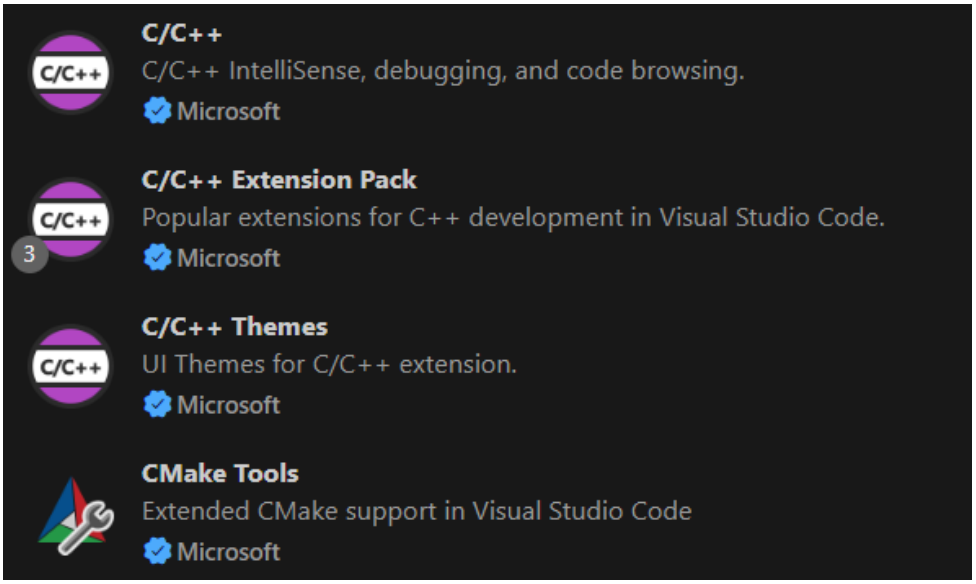

Figura 2: Extensões recomendadas no Visual Studio Code.

Informação

Observação importante:
O Visual Studio Code é apenas um editor de código. Mensagens de erro exibidas pelo IntelliSense (sublinhados vermelhos) não significam, necessariamente, erros reais de compilação.
O compilador utilizado neste tutorial é o `gcc` fornecido pelo MSYS2.

1.2 Instalação do MSYS2

1. Baixe o instalador do MSYS2 a partir do site oficial: `https://www.msys2.org/`

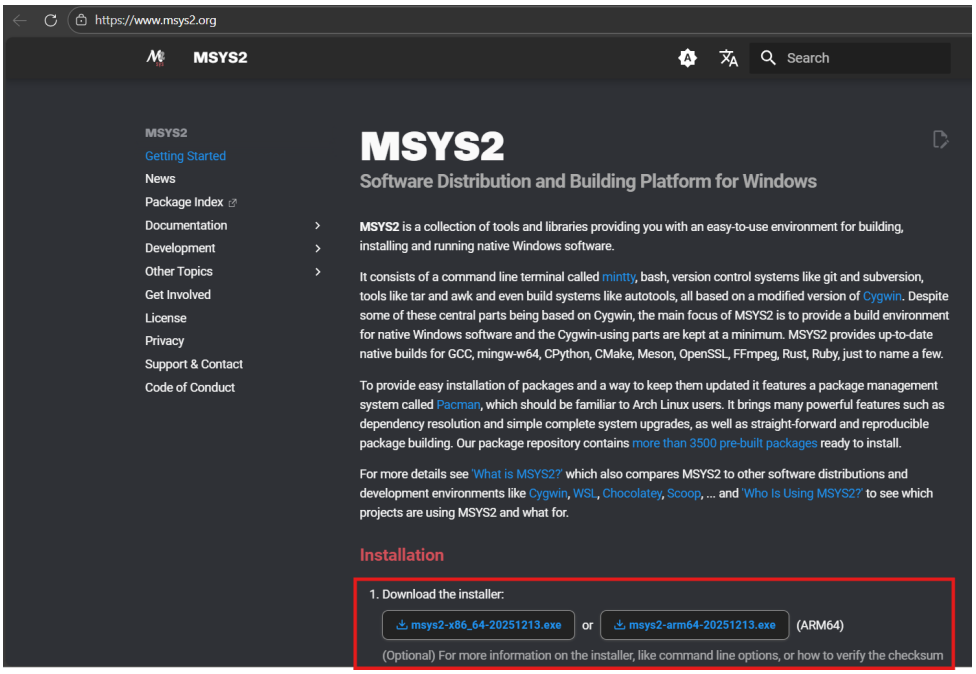

Figura 3: Página oficial de download do MSYS2.

2. Execute o instalador e siga as instruções padrão. Caso não haja necessidade específica, mantenha o diretório de instalação padrão: `C:\msys64`

3. Após a conclusão da instalação, abra o terminal **MSYS2 MINGW64** ou **MSYS2 UCRT64**. Ambos funcionam corretamente para este tutorial. **Não utilize o terminal MSYS2 básico.**

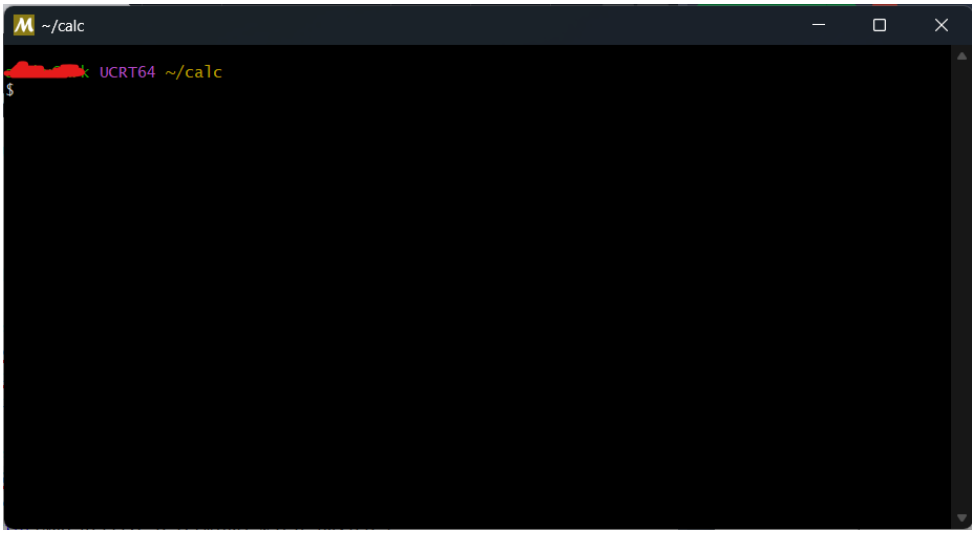

Figura 4: Terminal MSYS2 UCRT64.

Informação

Como localizar o terminal correto do MSYS2:

- Pressione a tecla Windows
- Digite: `MINGW64` ou `UCRT64`
- Selecione **MSYS2 MINGW64** ou **MSYS2 UCRT64** (ícone Azul ou Marrom respectivamente)

1.3 Instalação do Compilador GCC

No terminal **MSYS2 MINGW64** ou **UCRT64**, execute o comando abaixo para instalar o compilador C:

Comando para Copiar

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

```
resolving dependencies...
looking for conflicting packages...

Packages (15) mingw-w64-ucrt-x86_64-binutils-2.41-2
              mingw-w64-ucrt-x86_64-crt-git-11.0.0.r216.gffe883434-1
              mingw-w64-ucrt-x86_64-gcc-libs-13.2.0-2  mingw-w64-ucrt-x86_64-gmp-6.3.0-2
              mingw-w64-ucrt-x86_64-headers-git-11.0.0.r216.gffe883434-1
              mingw-w64-ucrt-x86_64-isl-0.26-1  mingw-w64-ucrt-x86_64-libiconv-1.17-3
              mingw-w64-ucrt-x86_64-libwinpthread-git-11.0.0.r216.gffe883434-1
              mingw-w64-ucrt-x86_64-mpc-1.3.1-2  mingw-w64-ucrt-x86_64-mpfr-4.2.1-2
              mingw-w64-ucrt-x86_64-windows-default-manifest-6.4-4
              mingw-w64-ucrt-x86_64-winpthreads-git-11.0.0.r216.gffe883434-1
              mingw-w64-ucrt-x86_64-zlib-1.3-1  mingw-w64-ucrt-x86_64-zstd-1.5.5-1
              mingw-w64-ucrt-x86_64-gcc-13.2.0-2

Total Download Size:    49.38 MiB
Total Installed Size:  418.82 MiB

:: Proceed with installation? [Y/n]
[... downloading and installation continues ...]
```

Figura 5: Instalação do compilador GCC utilizando o pacman.

Quando solicitado, digite `Y` e pressione Enter.

2 Estrutura do Projeto

2.1 Organização de Pastas

Crie a seguinte estrutura de diretórios para o projeto de testes unitários em C:

```
PROJETO_AUTO_TEST/
|-- inc/
|   |-- calc.h
|   |-- utest.h
|-- src/
|   |-- calc.c
|-- tests/
|   |-- test_calc.c
```

Listing 1: Estrutura do projeto

Informação

Significado de cada pasta:

- `inc/`: Arquivos de cabeçalho (`.h`), incluindo interfaces do projeto e o framework µTest.
- `src/`: Código de produção (implementação em C)
- `tests/`: Código de testes unitários

2.2 Download do µTest

O µTest é um framework de testes unitários **header-only**, ou seja, consiste em um único arquivo de cabeçalho.

1. Acesse o repositório oficial do µTest: `https://github.com/sheredom/utest.h`
2. Clique em **Code** → **Download ZIP** ou baixe diretamente o arquivo `utest.h`.
3. Extraia o arquivo `utest.h`.
4. Copie o arquivo `utest.h` para a pasta `inc/` de seu projeto.

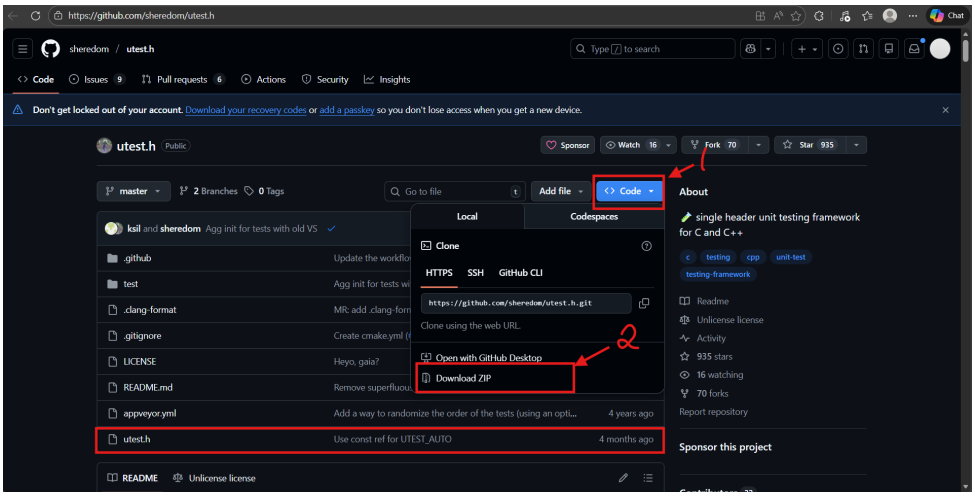

Figura 6: Download do arquivo `utest.h` no GitHub.

Sucesso

O uTest não requer instalação adicional, compilação separada ou bibliotecas externas. Basta incluir o arquivo `utest.h` no projeto para começar a escrever testes.

3 Código-Fonte

Esta seção apresenta o código-fonte do projeto, incluindo a interface da calculadora, sua implementação em C e a suíte de testes unitários utilizando o framework uTest.

3.1 Header: `calc.h`

Crie o arquivo `inc/calc.h`. Este arquivo define a interface pública da biblioteca de cálculo, incluindo códigos de erro e protótipos das funções.

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

#include <stdbool.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * C digos de erro retornados pelas fun   es
 */
typedef enum
{
    CALC_OK = 0,
    CALC_ERROR_DIV_BY_ZERO,
    CALC_ERROR_INVALID_ARGUMENT
} calc_status_t;

```

```c
/*
 * Prot tipos das fun   es da calculadora
 */
calc_status_t calculator_add(double a, double b, double *result);
calc_status_t calculator_sub(double a, double b, double *result);
calc_status_t calculator_mul(double a, double b, double *result);
calc_status_t calculator_div(double a, double b, double *result);

calc_status_t calculator_pow(double base, double exp, double *
   result);
calc_status_t calculator_mod(int a, int b, int *result);
calc_status_t calculator_abs(double a, double *result);
calc_status_t calculator_min(double a, double b, double *result);
calc_status_t calculator_max(double a, double b, double *result);

#ifdef __cplusplus
}
#endif

#endif /* CALCULATOR_H */
```

Listing 2: inc/calc.h

Informação

Observações sobre o header:

- A biblioteca é escrita em C, mas compatível com C++ via `extern "C"`
- Todas as funções retornam um código de status
- O resultado das operações é retornado via ponteiro

3.2 Implementação: `calc.c`

Crie o arquivo `src/calc.c`. Este arquivo contém a implementação das funções declaradas no header.

```c
#include "../inc/calc.h"
#include <math.h>

/* Fun   o auxiliar para valida   o de ponteiros */
static bool is_null(void *ptr)
{
    return ptr == NULL;
}

/* ================================
 * Operacoes basicas
 * ================================ */
calc_status_t calculator_add(double a, double b, double *result)
{
```

```
    if (is_null(result)) {
        return CALC_ERROR_INVALID_ARGUMENT;
    }

    *result = a + b;
    return CALC_OK;
}

calc_status_t calculator_sub(double a, double b, double *result)
{
    if (is_null(result)) {
        return CALC_ERROR_INVALID_ARGUMENT;
    }

    *result = a - b;
    return CALC_OK;
}

calc_status_t calculator_mul(double a, double b, double *result)
{
    if (is_null(result)) {
        return CALC_ERROR_INVALID_ARGUMENT;
    }

    *result = a * b;
    return CALC_OK;
}

calc_status_t calculator_div(double a, double b, double *result)
{
    if (is_null(result)) {
        return CALC_ERROR_INVALID_ARGUMENT;
    }

    if (b == 0.0) {
        return CALC_ERROR_DIV_BY_ZERO;
    }

    *result = a / b;
    return CALC_OK;
}

/* ================================
 * Operacoes avancadas
 * ================================ */
calc_status_t calculator_pow(double base, double exp, double *
    result)
{
    if (is_null(result)) {
        return CALC_ERROR_INVALID_ARGUMENT;
    }
```

```
65
66      *result = pow(base, exp);
67      return CALC_OK;
68  }
69
70  calc_status_t calculator_mod(int a, int b, int *result)
71  {
72      if (is_null(result)) {
73          return CALC_ERROR_INVALID_ARGUMENT;
74      }
75
76      if (b == 0) {
77          return CALC_ERROR_DIV_BY_ZERO;
78      }
79
80      *result = a % b;
81      return CALC_OK;
82  }
83
84  calc_status_t calculator_abs(double a, double *result)
85  {
86      if (is_null(result)) {
87          return CALC_ERROR_INVALID_ARGUMENT;
88      }
89
90      *result = (a < 0.0) ? -a : a;
91      return CALC_OK;
92  }
93
94  calc_status_t calculator_min(double a, double b, double *result)
95  {
96      if (is_null(result)) {
97          return CALC_ERROR_INVALID_ARGUMENT;
98      }
99
100     *result = (a < b) ? a : b;
101     return CALC_OK;
102 }
103
104 calc_status_t calculator_max(double a, double b, double *result)
105 {
106     if (is_null(result)) {
107         return CALC_ERROR_INVALID_ARGUMENT;
108     }
109
110     *result = (a > b) ? a : b;
111     return CALC_OK;
112 }
```

Listing 3: src/calc.c

3.3 Testes Unitários: `test_calc.c`

Os testes unitários são implementados utilizando o framework **uTest**. Cada função da calculadora possui testes para:

- Casos de sucesso
- Parâmetros inválidos
- Condições de erro (ex: divisão por zero)

Crie o arquivo `tests/test_calc.c`:

```c
#include "../inc/utest.h"
#include "../inc/calc.h"

/* Tolerancia para comparacao de ponto flutuante */
#define TOLERANCE 1e-6

UTEST(calculator_add, success)
{
    double result = 0.0;
    ASSERT_EQ(CALC_OK, calculator_add(2.0, 3.0, &result));
    ASSERT_NEAR(5.0, result, TOLERANCE);
}

UTEST(calculator_add, null_result)
{
    ASSERT_EQ(CALC_ERROR_INVALID_ARGUMENT,
              calculator_add(1.0, 2.0, NULL));
}

UTEST(calculator_div, divide_by_zero)
{
    double result = 0.0;
    ASSERT_EQ(CALC_ERROR_DIV_BY_ZERO,
              calculator_div(10.0, 0.0, &result));
}

/* Funcao main dos testes */
UTEST_MAIN();
```

Listing 4: tests/test_calc.c (exemplo basico)

Informação

Entendendo o código de teste com uTest:

- `UTEST(grupo, nome)`: Define um caso de teste
- `ASSERT_EQ`: Verifica igualdade
- `ASSERT_NEAR`: Verifica valores de ponto flutuante
- `UTEST_MAIN()`: Gera automaticamente a função `main`

4 Fluxo de Trabalho (Build e Testes)

4.1 Workflow Completo

O fluxo de trabalho recomendado para desenvolvimento e testes neste projeto é o seguinte:

1. **Editar o código:** Utilize o Visual Studio Code para modificar os arquivos `.c` e `.h`.
2. **Salvar as alterações:** Utilize `Ctrl + S` no VS Code.
3. **Alternar para o MSYS2:** Use `Alt + Tab` ou selecione a janela do terminal.
4. **Compilar o projeto:** Execute os comandos de compilação no terminal MSYS2.
5. **Executar os testes:** Rode o executável de testes.
6. **Analisar os resultados:** Verifique a saída gerada pelo µTest.
7. **Retornar ao VS Code:** Faça os ajustes necessários no código.

4.2 Navegação no MSYS2

Abra o terminal **MSYS2 MINGW64** ou **MSYS2 UCRT64** e navegue até o diretório raiz do projeto:

Comando para Copiar

```
cd /c/PROJETO_EXEMPLO
```

Verifique se o diretório atual está correto:

Comando para Copiar

```
pwd
```

Liste os arquivos e pastas do projeto:

Comando para Copiar

```
ls
```

Atenção

Sintaxe correta de caminhos no MSYS2:

- Windows: `C:\PROJETO_EXEMPLO`
- MSYS2: `/c/PROJETO_EXEMPLO`

Utilize barras normais (`/`) e letras minúsculas para identificar o drive.

4.3 Compilação do Projeto

Para compilar o código de produção junto com a suíte de testes, execute:

Comando para Copiar

```
gcc -std=c99 -Wall -o test_calc
src/*.c tests/*.c -I./inc -lm
```

Informação

Explicação do comando de compilação:

- `gcc`: Compilador da linguagem C fornecido pelo MSYS2
- `-std=c99`: Define o padrão da linguagem C como C99
- `-Wall`: Habilita os principais avisos do compilador para ajudar a identificar possíveis erros no código
- `-o test_calc`: Define o nome do arquivo executável gerado
- `src/*.c`: Inclui todos os arquivos `.c` do diretório de código de produção
- `tests/*.c`: Inclui todos os arquivos `.c` do diretório de testes unitários
- `-I./inc`: Informa ao compilador onde localizar os arquivos de cabeçalho (`.h`)
- `-lm`: Vincula a biblioteca matemática padrão, necessária para funções como `pow()`

Observação: O caractere `*` é um curinga que inclui todos os arquivos `.c` do diretório especificado.

Sucesso

Compilação concluída com sucesso:
Se nenhum erro for exibido, o executável `test_calc.exe` será gerado no diretório raiz do projeto.

Erro Comum

Erros comuns durante a compilação:

- `No such file or directory`: Caminho incorreto ou arquivo inexistente
- `undefined reference`: Arquivo `.c` não incluído na compilação ou biblioteca ausente
- `utest.h: No such file or directory`: Verifique o diretório informado em `-I`

4.4 Execução dos Testes

Após a compilação, execute a suíte de testes:

Comando para Copiar

```
./test_calc
```

4.4.1 Saída Esperada

```
[==========] Running tests...
[----------] All tests passed
[==========] Done
```

Listing 5: Exemplo de saída do uTest

4.5 Comando Combinado (Build + Testes)

É possível compilar e executar os testes em um único comando:

Comando para Copiar

```
gcc -std=c99 -Wall -o test_calc
src/*.c tests/*.c -I./inc -lm && ./test_calc
```

Informação

O operador `&&` indica que os testes serão executados apenas se a compilação for concluída com sucesso.

5 Comandos Avançados do µTest

O µTest possui uma proposta simples e minimalista. Diferente de frameworks mais complexos, ele não utiliza opções de linha de comando para filtragem, geração de relatórios ou modos verbosos. Toda a configuração é feita diretamente no código-fonte dos testes.

5.1 Execução dos Testes

Para executar todos os testes definidos no projeto, utilize:

Comando para Copiar

```
./test_calc
```

Este comando executa automaticamente todos os casos de teste registrados no programa.

Informação

Importante: O µTest executa sempre todos os testes compilados no binário. Não há necessidade de parâmetros adicionais para ativar ou desativar testes.

5.2 Controle de Saída

A saída padrão do uTest já informa:

- Início da execução
- Resultado de cada teste
- Indicação clara de falhas
- Status final da suíte

Exemplo de saída típica:

```
[==========] Running tests...
[----------] All tests passed
[==========] Done
```

Listing 6: Saída padrão do uTest

Informação

Em caso de falha, o µTest informa exatamente:

- Qual teste falhou
- Arquivo e linha do erro
- Expressão que falhou

6 Dicas e Boas Práticas

6.1 Organização dos Testes

- Crie um arquivo de teste separado para cada módulo do sistema
- Use funções de teste pequenas e focadas
- Teste apenas uma funcionalidade por teste
- Evite dependência entre testes

6.2 Nomenclatura

- Use nomes claros e descritivos para os testes
- Mantenha consistência no idioma (português ou inglês)
- Prefira nomes que indiquem comportamento esperado

Exemplo:

```
UTEST(calc, addition_positive_numbers) {
    ASSERT_EQ(5, calc_add(2, 3));
}
```

7 Troubleshooting

7.1 Problemas Comuns

Erro Comum

Erro: "gcc: command not found"
Solução: Você não está utilizando o terminal correto. Abra o **MSYS2 MINGW64** ou **MSYS2 UCRT64**.

Erro Comum

Erro: "No such file or directory"
Solução:

- Confirme que você está na pasta raiz do projeto (`pwd`)
- Verifique os nomes dos diretórios (`src`, `tests`, `inc`)
- Utilize `ls` para conferir os arquivos existentes

Erro Comum

Erro: "undefined reference"
Solução: Algum arquivo `.c` não foi incluído no comando de compilação. Verifique se todos os arquivos de `src/` estão sendo compilados.

Erro Comum

Erro: "utest.h: No such file or directory"
Solução:

- Confirme que `utest.h` está dentro da pasta `inc/`
- Verifique se o parâmetro `-I./inc` está presente no comando

8 Conclusão

Neste tutorial foi apresentada uma abordagem prática e confiável para testes unitários em C no Windows:

- **Visual Studio Code:** Editor leve e eficiente para desenvolvimento
- **MSYS2 MINGW64/UCRT64:** Ambiente robusto para compilação
- **µTest:** Framework simples, rápido e ideal para projetos em C

8.1 Recursos Adicionais

- Repositório oficial do µTest: `https://github.com/sheredom/utest.h`
- Documentação do MSYS2: `https://www.msys2.org/docs/`
- Boas práticas em testes de software: `https://martinfowler.com/testing/`
- Boas práticas de testes C/C++: `https://github.com/isocpp/CppCoreGuidelines`