Module: Fundamentals of Test Automation
Student: Leonardo de Melo Abreu
Professor: Alexandro Ferreira de Miranda

Link for the video: https://drive.google.com/file/d/1_sRAHy4WWRPJnl_ZNNdhKaNn-raE9j-T/view?usp=sharing

Link for GitHub: https://github.com/leonardomello27/Tutorial_CppUnitTest/tree/main

Tutorial

Overview

In this tutorial, we will learn about unit testing, its importance, and how to create tests in Visual Studio 2022. We will also explore practical examples of creating tests to ensure the correct functionality of the code for a calculator.

What are Unit Tests?

At its core, unit tests are a powerful tool that allows us to thoroughly examine specific parts of our code. Think of them as magnifying lenses that focus on libraries, classes, functions, and various elements of the program. The underlying goal is to determine if the code is not only operational but also operating accurately and as expected for a well-defined set of inputs.

Unit tests go beyond merely checking if the software works. They are designed to explore the nooks and crannies of functionalities, exposing any discrepancies between actual behavior and the desired one. By focusing on distinct code fragments, we ensure that each part contributes to the overall functionality, preventing slip-ups that could go unnoticed in broader tests.

CppUnitTest

CppUnitTest is a unit testing tool for C++ that is part of Visual Studio. It provides a framework for writing and executing unit tests, as well as visualizing test results.

Microsoft Visual Studio 2022

Considering that CppUnitTest is present in the Microsoft Visual Studio 2022 environment, we will walk through the step-by-step process of creating unit tests using this tool. The intention is for this tutorial to illustrate concepts and demonstrate the necessary configurations for developing unit tests through Microsoft Visual Studio 2022.

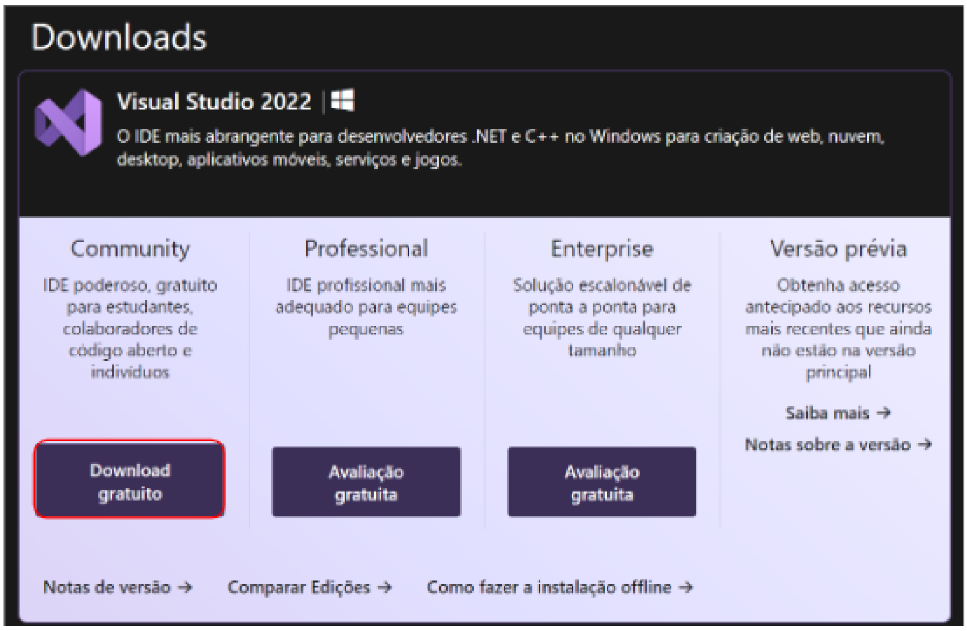
Getting Started

Step 1 -Install Microsoft Visual Studio 2022.

Disponível em: https://visualstudio.microsoft.com/pt-br/downloads/

Available at: https://visualstudio.microsoft.com/en/downloads/

The software can be downloaded for free by clicking the "Free Download" button.



Downloads

Visual Studio 2022

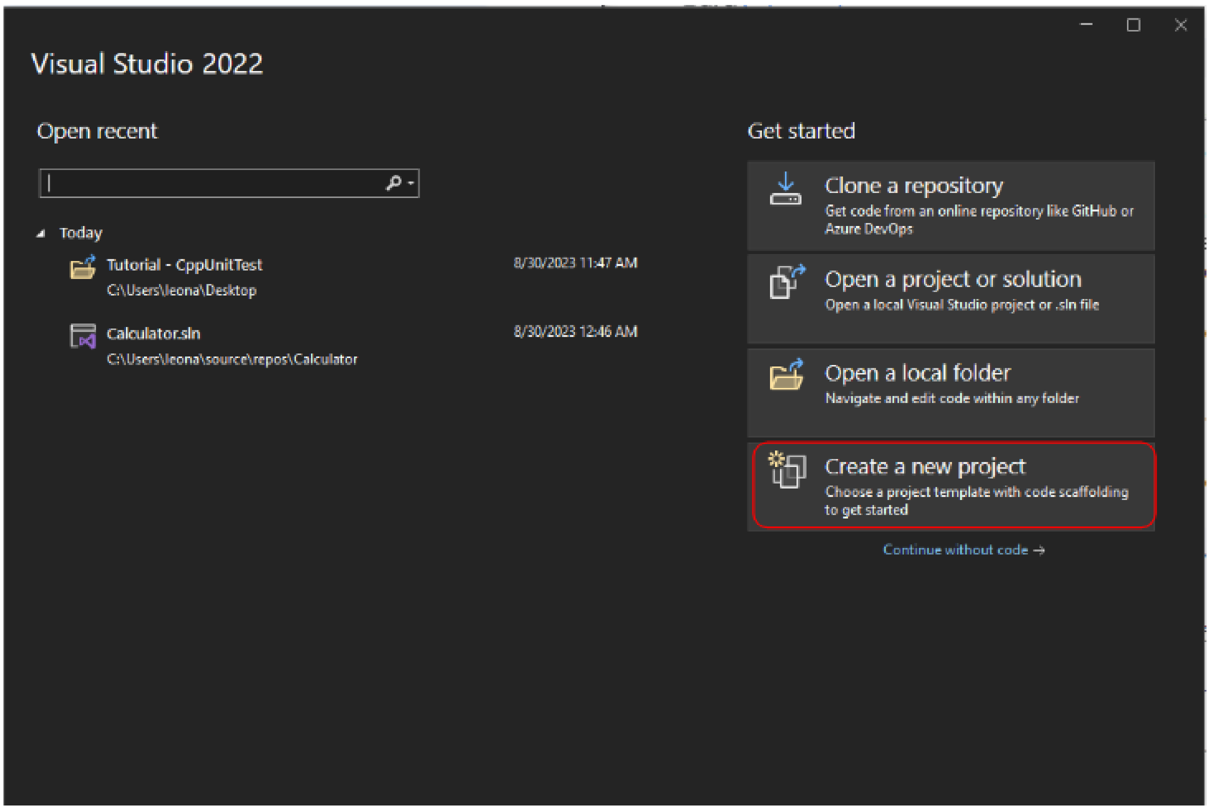
O IDE mais abrangente para desenvolvedores .NET e C++ no Windows para criação de web, nuvem, desktop, aplicativos móveis, serviços e jogos.

Community	Professional	Enterprise	Versão prévia
IDE poderoso, gratuito para estudantes, colaboradores de código aberto e indivíduos	IDE profissional mais adequado para equipes pequenas	Solução escalonável de ponta a ponta para equipes de qualquer tamanho	Obtenha acesso antecipado aos recursos mais recentes que ainda não estão na versão principal
Download gratuito	Avaliação gratuita	Avaliação gratuita	Saiba mais → Notas sobre a versão →

Notas de versão → Comparar Edições → Como fazer a instalação offline →

Note: The installation process is quite straightforward; you simply need to run the downloaded executable file and follow the recommended installation instructions.

Step 2 - With Microsoft Visual Studio 2022 already installed, you will need to create a new project. To do this, open Microsoft Visual Studio 2022. On the start page, click on "Create a new project."


Visual Studio 2022

Open recent

Today

Tutorial - CppUnitTest
C:\Users\leona\Desktop
8/30/2023 11:47 AM

Calculator.sln
C:\Users\leona\source\repos\Calculator
8/30/2023 12:46 AM

Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

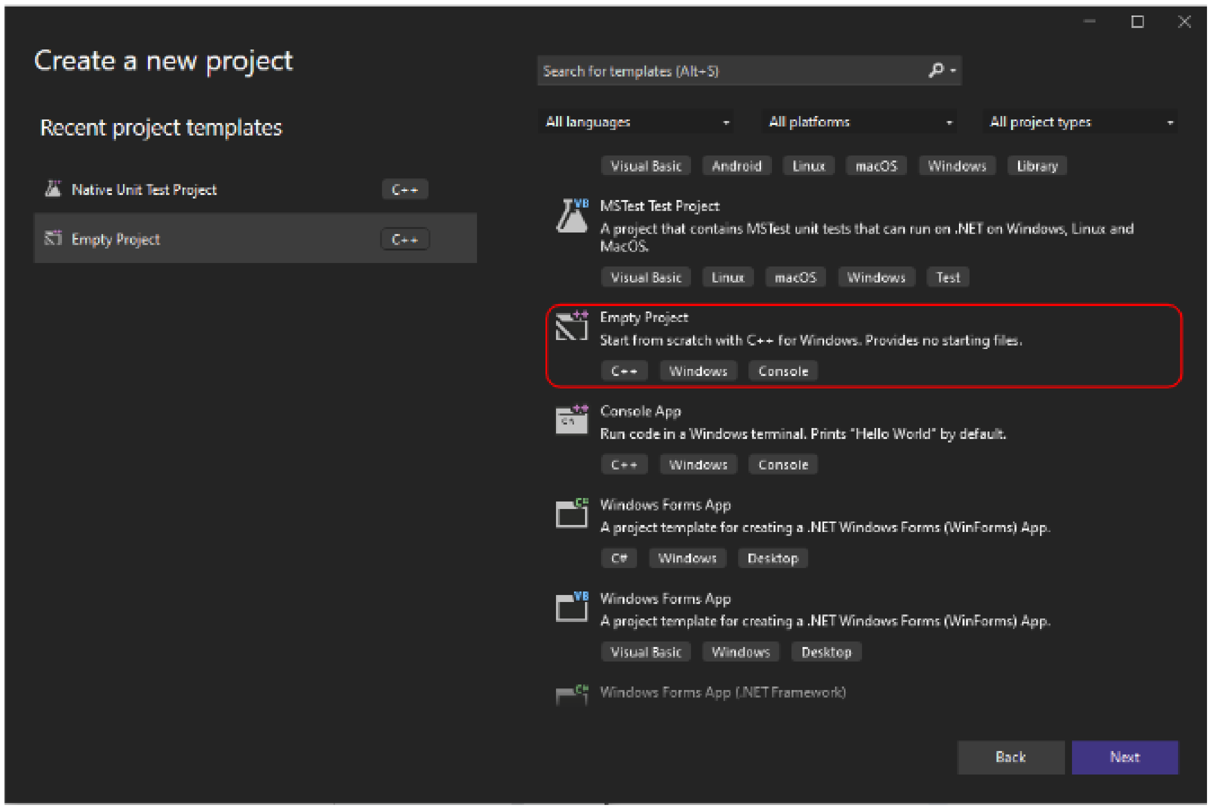
Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

Then, select the "Empty Project" option in the tab that opens, and click the "Next" button.


Create a new project

Recent project templates

Native Unit Test Project C++

Empty Project C++

Search for templates (Alt+S)

All languages · All platforms · All project types

Visual Basic · Android · Linux · macOS · Windows · Library

MSTest Test Project
A project that contains MSTest unit tests that can run on .NET on Windows, Linux and MacOS.
Visual Basic · Linux · macOS · Windows · Test

Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ · Windows · Console

Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ · Windows · Console

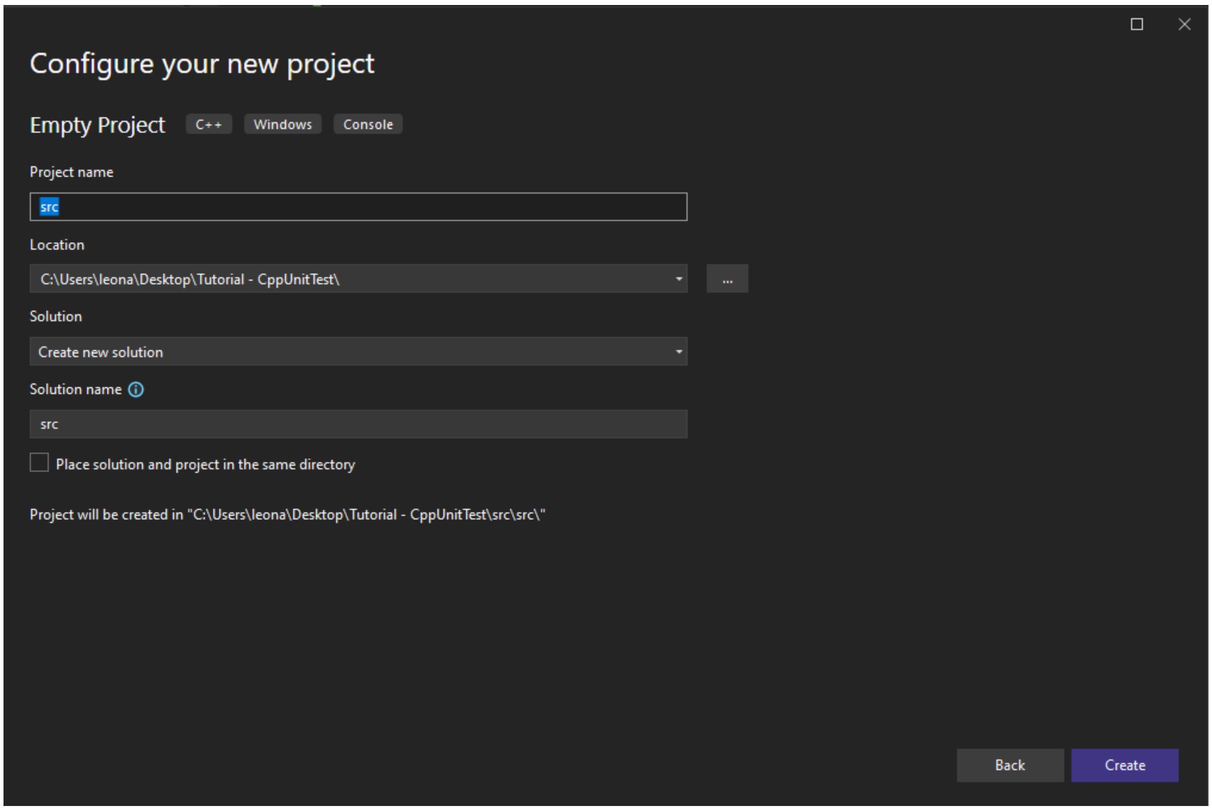
Windows Forms App
A project template for creating a .NET Windows Forms (WinForms) App.
C# · Windows · Desktop

Windows Forms App
A project template for creating a .NET Windows Forms (WinForms) App.
Visual Basic · Windows · Desktop

Windows Forms App (.NET Framework)

Back · Next

Now, simply enter the project name in the "Project name" field and click the "Create" button.


Configure your new project

Empty Project C++ Windows Console

Project name: src

Location: C:\Users\leona\Desktop\Tutorial - CppUnitTest\

Solution: Create new solution

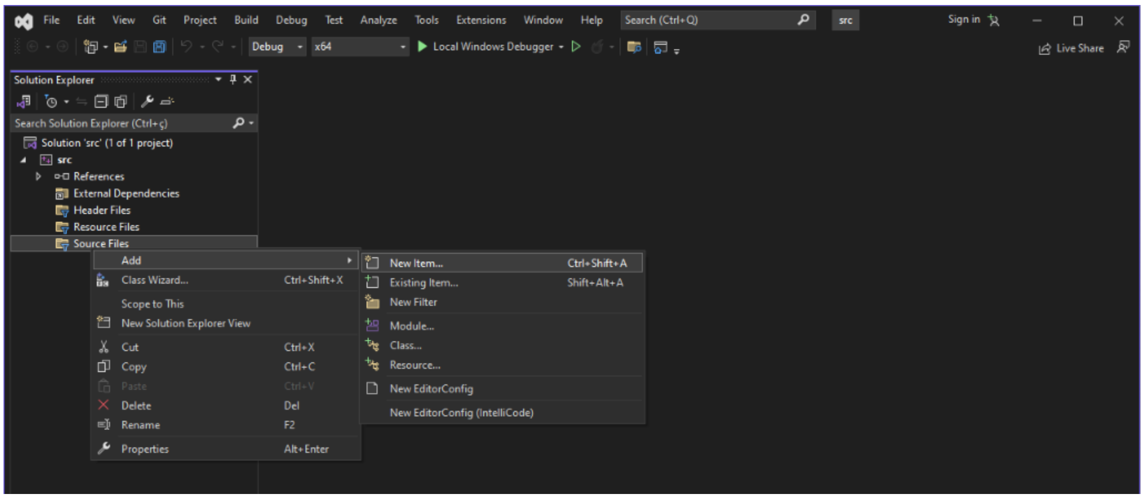
Solution name: src

☐ Place solution and project in the same directory

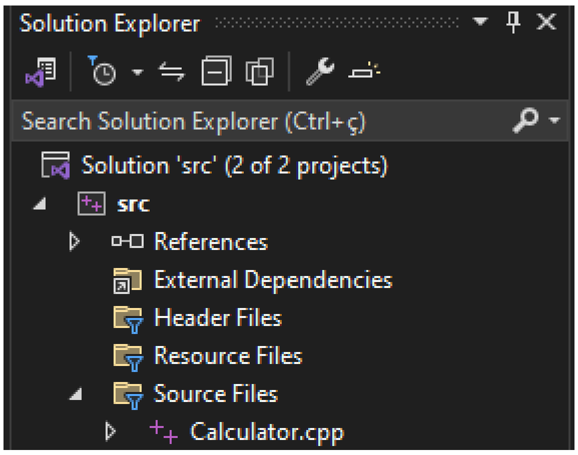
Project will be created in "C:\Users\leona\Desktop\Tutorial - CppUnitTest\src\src\"

Step 3 - Now that we have created the project, we need to create the file that will contain the code we will use for our tests.

To do this, right-click on the "Source Files" folder. When you do this, a new window will open with several options. Select the "Add" option, and then click on "New Item."



Now you can see the file with the chosen name in the "Solution Explorer" menu, along with the project folders. For this example, a file named "Calculator.cpp" has been created.

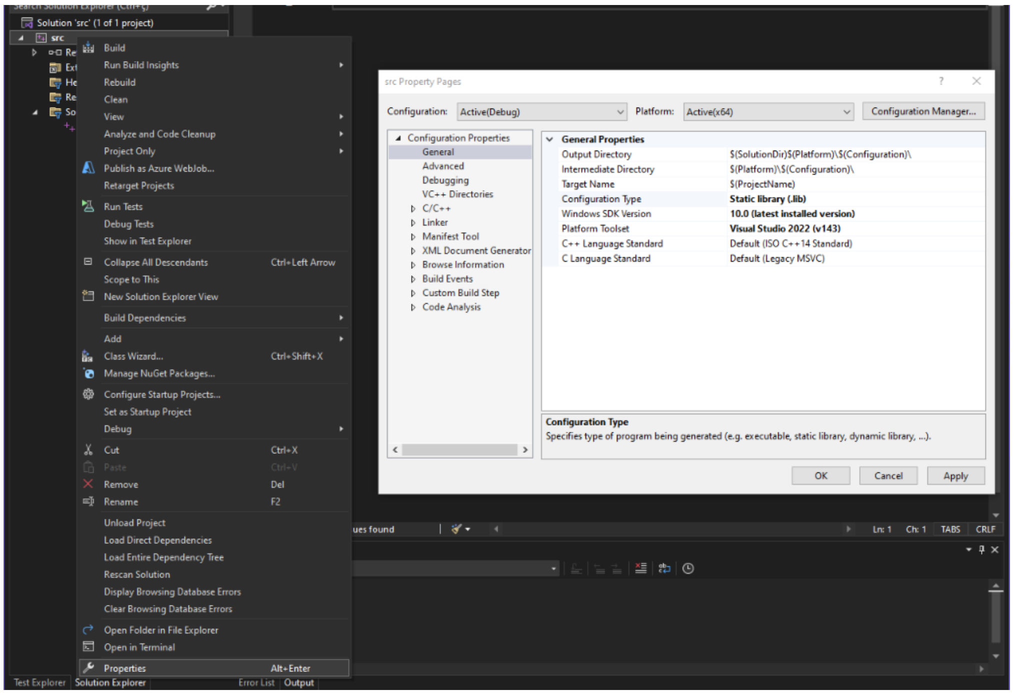


Clicking on the "Calculator.cpp" file will open the text editor for typing in the code. In the following section, you'll find the example code that was used as the basis for this tutorial:

```
#include <iostream>
#include <stdexcept>

int soma(int a, int b) {
    return a + b;
}

int sub(int a, int b) {
    return a - b;
}

int mult(int a, int b) {
    return a * b;
}

int divide(int a, int b)
{
    if (b == 0)
    {

        throw std::invalid_argument("Divisão por zero."); // Lança uma exceção
    }

    return a / b;
}
```

The next step to take is to configure the "Calculator.cpp" file as a library so that we can easily use it to test the functions implemented in the project that will be created later with the test cases.

You can do this by right-clicking on the project name, in this case, "src". A menu with various options will appear; click on "Properties." A window will open. Click on "Configuration Properties" -> "General" and change the "Configuration Type" to "Static library (.lib)" and click "Apply".



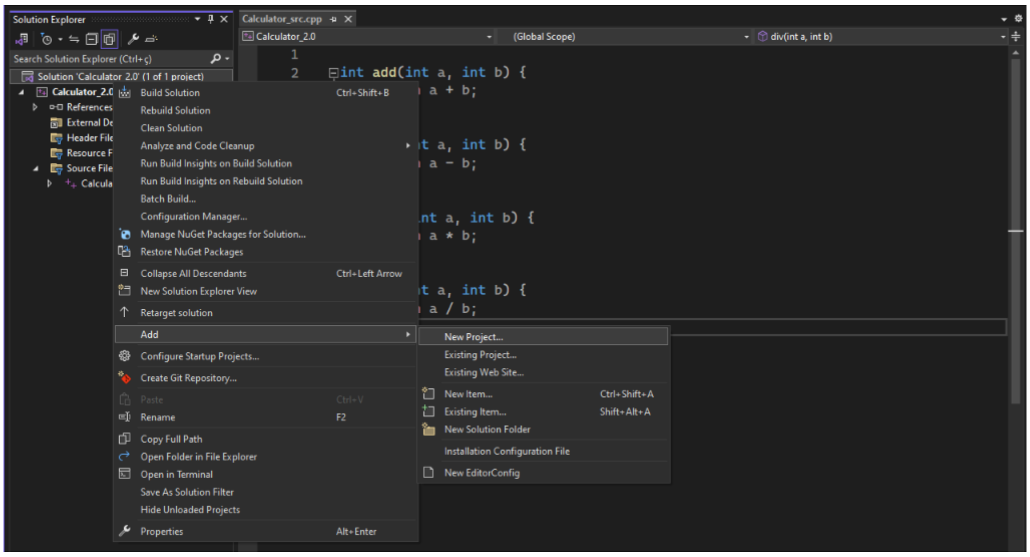
Configuration: Active(Debug)   Platform: Active(x64)

General Properties

Output Directory	$(SolutionDir)$(Platform)\$(Configuration)\
Intermediate Directory	$(Platform)\$(Configuration)\
Target Name	$(ProjectName)
Configuration Type	**Static library (.lib)**
Windows SDK Version	**10.0 (latest installed version)**
Platform Toolset	**Visual Studio 2022 (v143)**
C++ Language Standard	Default (ISO C++14 Standard)
C Language Standard	Default (Legacy MSVC)

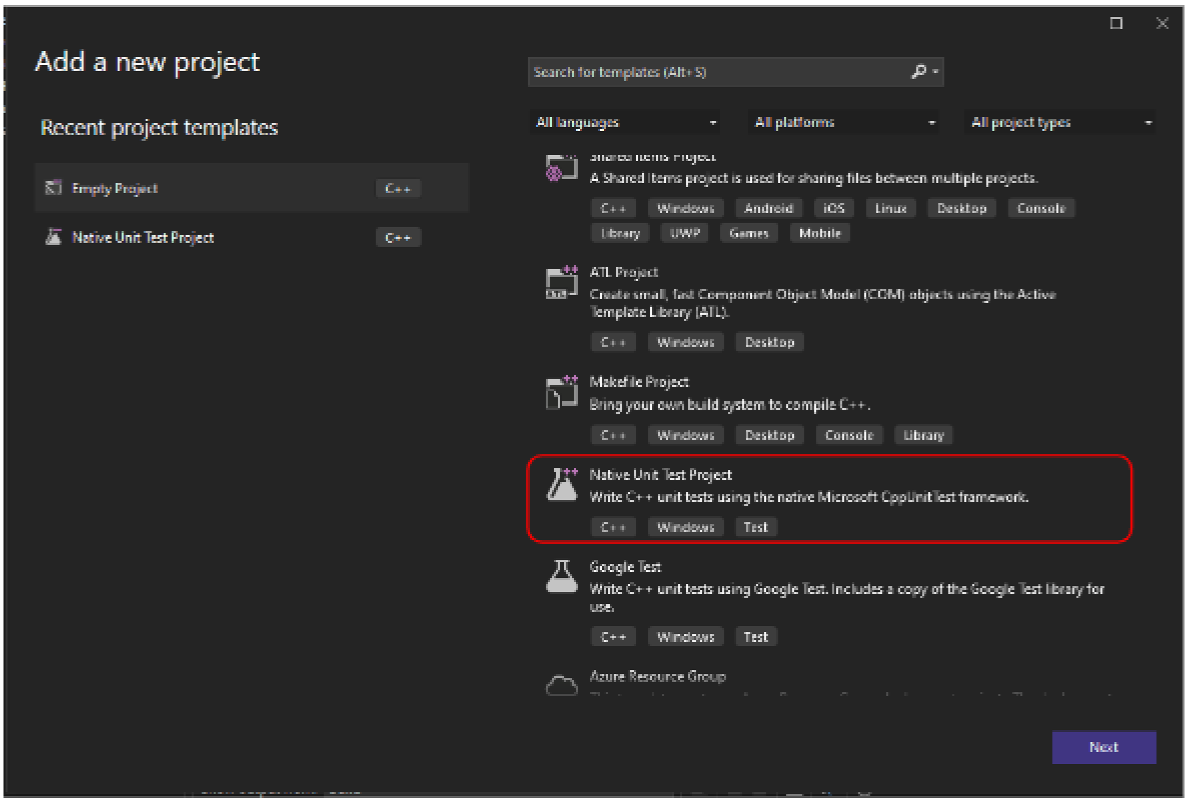
Configuration Type
Specifies type of program being generated (e.g. executable, static library, dynamic library, ...).

Step 4 - Great! Now that we have the first project set up, let's create the project that will be responsible for containing the test cases.
To do that, click on "Solution <project name>." An auxiliary tab will open. Click on the "Add" option -> "New Project".



```
int add(int a, int b) {
  a + b;

t a, int b) {
  a - b;

nt a, int b) {
  a * b;

t a, int b) {
  a / b;
```

When you click on "New Project," the following window will open:



Add a new project

Search for templates (Alt+ S)

Recent project templates

- Empty Project — C++
- Native Unit Test Project — C++

All languages · All platforms · All project types

Shared Items Project
A Shared Items project is used for sharing files between multiple projects.
C++ · Windows · Android · iOS · Linux · Desktop · Console · Library · UWP · Games · Mobile

ATL Project
Create small, fast Component Object Model (COM) objects using the Active Template Library (ATL).
C++ · Windows · Desktop

Makefile Project
Bring your own build system to compile C++.
C++ · Windows · Desktop · Console · Library

Native Unit Test Project
Write C++ unit tests using the native Microsoft CppUnitTest framework.
C++ · Windows · Test

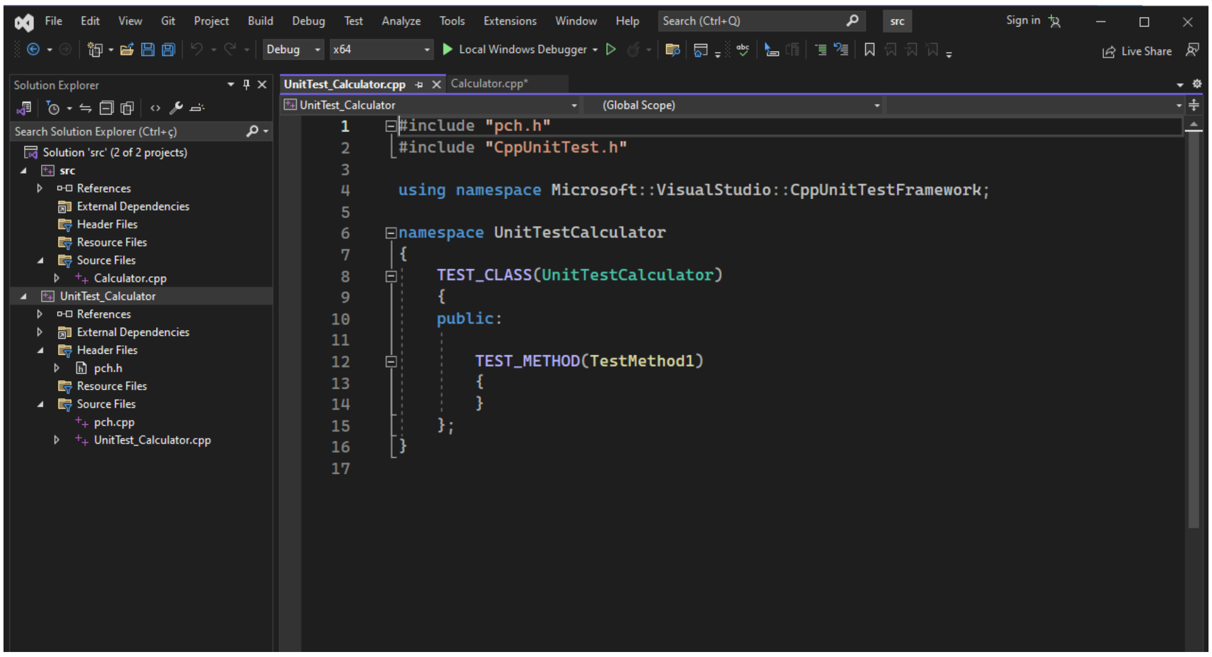
Google Test
Write C++ unit tests using Google Test. Includes a copy of the Google Test library for use.
C++ · Windows · Test

Azure Resource Group

Next

In this window, we will create a project for unit testing using Microsoft's CppUnitTest framework.

In the following figure, you can see that the project has been created successfully. When the project is created, the tool itself generates a structure that serves as a foundation for test development.



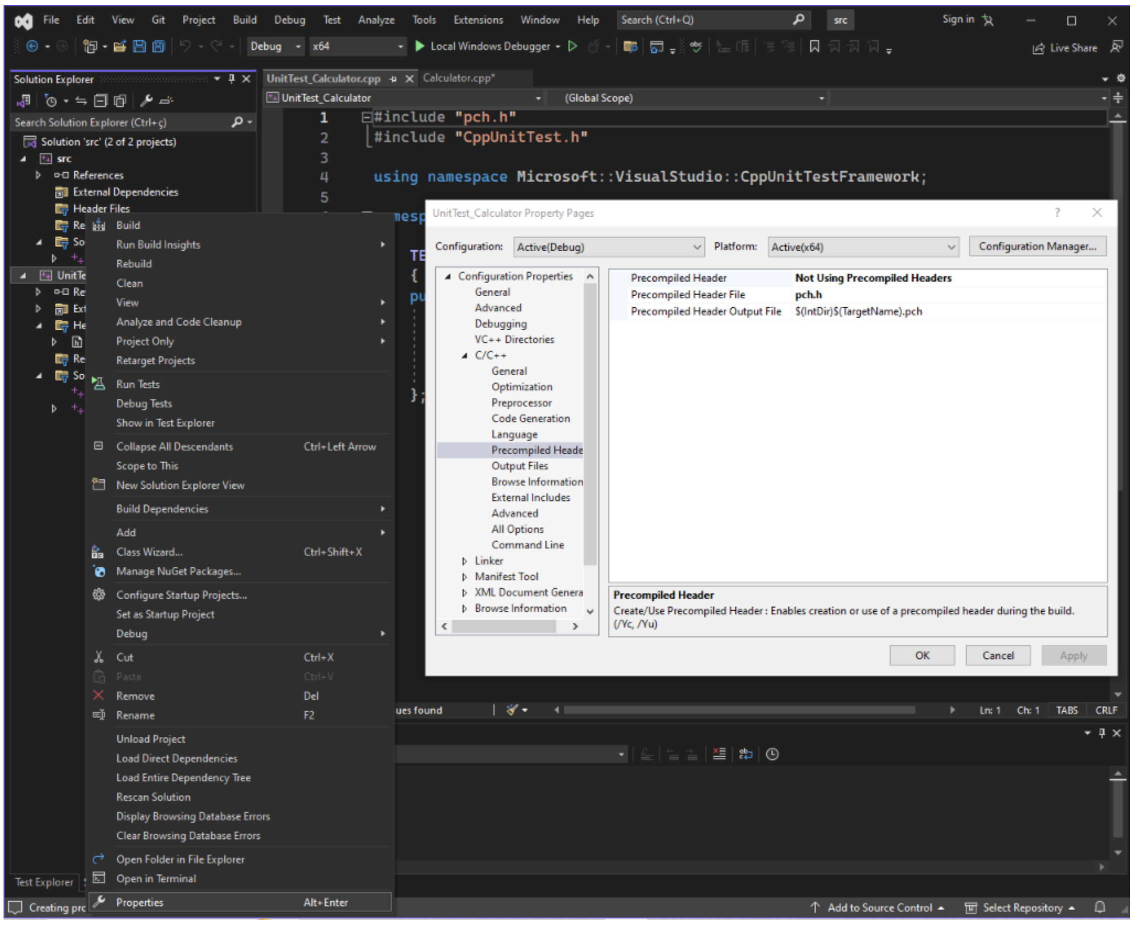
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
src
Sign in
Debug · x64 · Local Windows Debugger
Live Share
UnitTest_Calculator.cpp · Calculator.cpp*
UnitTest_Calculator · (Global Scope)

```
Solution Explorer
Search Solution Explorer (Ctrl+ç)
Solution 'src' (2 of 2 projects)
  src
    References
    External Dependencies
    Header Files
    Resource Files
    Source Files
      Calculator.cpp
  UnitTest_Calculator
    References
    External Dependencies
    Header Files
      pch.h
    Resource Files
    Source Files
      pch.cpp
      UnitTest_Calculator.cpp
```

```cpp
#include "pch.h"
#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestCalculator
{
    TEST_CLASS(UnitTestCalculator)
    {
    public:

        TEST_METHOD(TestMethod1)
        {
        }
    };
}
```

Upon creating the project, you'll notice that "php.cpp" and "php.h" files are generated. These are default files that provide a basic structure to start writing code. However, we won't be using them. Hence, you can delete these files and request the system not to use these precompiled files.

To do this, right-click on the name of the test project. A tab will open; click on "Properties" -> "Configuration Properties" -> "Precompiled Header," and then select "<Not Using Precompiled Headers>" in the "Precompiled Header" field.



```
#include "pch.h"
#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;
```

UnitTest_Calculator Property Pages

Configuration: Active(Debug) Platform: Active(x64)

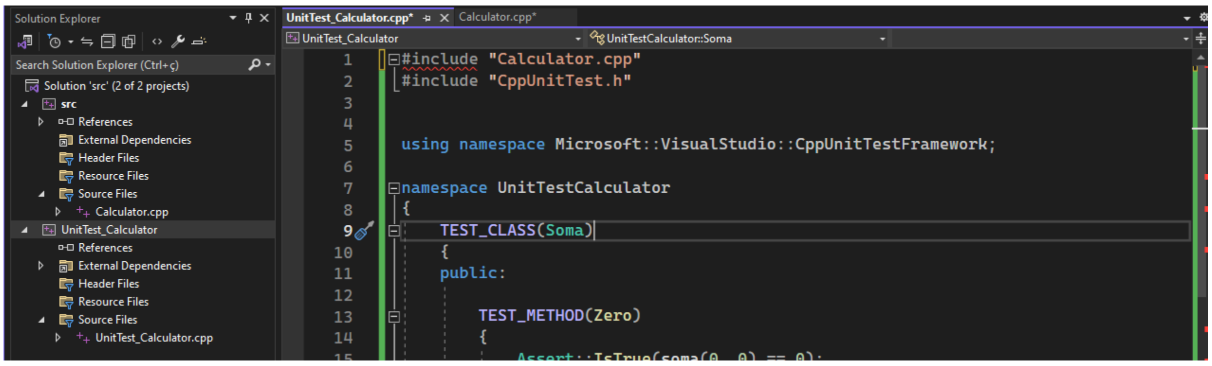
Precompiled Header	Not Using Precompiled Headers
Precompiled Header File	pch.h
Precompiled Header Output File	$(IntDir)$(TargetName).pch

Precompiled Header
Create/Use Precompiled Header : Enables creation or use of a precompiled header during the build. (/Yc, /Yu)

Step 5 - With the file configured, you can now start writing the test cases. For this tutorial, the following test cases were written:

```
#include "Calculator.cpp"      // Inclui o arquivo de implementação da calculadora
#include "CppUnitTest.h"       // Inclui a biblioteca do CppUnitTestFramework

using namespace Microsoft::VisualStudio::CppUnitTestFramework; // Espaço de nomes para asserções do
//Visual Studio

namespace UnitTestCalculator // Namespace para os testes da calculadora
{
        TEST_CLASS(Soma) // Classe de testes para a função de soma
        {
        public:

                TEST_METHOD(Zero)               // Método de teste para a soma com valores zero
                {
                        Assert::IsTrue(soma(0, 0) == 0);  // Verifica se a soma de 0 + 0 é igual a 0
                        Assert::IsTrue(soma(0, 1) == 1);  // Verifica se a soma de 0 + 1 é igual a 1
                }
                TEST_METHOD(Básico)                  // Método de teste para somas básicas
                {
                        Assert::IsTrue(soma(2, 5) == 7);  // Verifica se a soma de 2 + 5 é igual a 7
                        Assert::IsTrue(soma(-2, 5) == 3); // Verifica se a soma de -2 + 5 é igual a 3

                        Assert::AreEqual(soma(2, 5), 7);  // Verifica se a soma de 2 + 5 é igual a 7
                        Assert::AreEqual(soma(-2, 5), 3); // Verifica se a soma de -2 + 5 é igual a 3
                }
                TEST_METHOD(Comutativa)   // Método de teste para a propriedade comutativa da soma
                {
                        Assert::AreEqual(soma(2, 5), soma(5, 2)); // Verifica se a soma de 2 + 5 é igual à
//soma de 5 + 2
                }
                TEST_METHOD(Fal)                // Método de teste para verificar falso
                {
                        Assert::IsFalse(soma(0, 0) == 1); // Verifica se a soma de 0 + 0 não é igual a 1
                }

        };

        TEST_CLASS(Subitração) // Classe de testes para a função de subtração
        {
        public:

                TEST_METHOD(Zero)                   // Método de teste para subtrações com valor zero
                {
                        Assert::IsTrue(sub(0, 0) == 0);  // Verifica se a subtração de 0 - 0 é igual a 0
                        Assert::AreEqual(sub(0, 5), -5); // Verifica se a subtração de 0 - 5 é igual a -5
                }

                TEST_METHOD(Básico)                  // Método de teste para subtrações básicas
                {
                        Assert::IsTrue(sub(3, 4) == -1); // Verifica se a subtração de 3 - 4 é igual a -1
                        Assert::IsTrue(sub(-2, 5) == -7);// Verifica se a subtração de -2 - 5 é igual a -7
                }

                TEST_METHOD(False)               // Método de teste para verificar falso
                {
                        Assert::IsFalse(sub(0, 0) == 1); // Verifica se a subtração de 0 - 0 não é igual a 1
                }

                TEST_METHOD(SelfSub)               // Método de teste para subtração consigo mesmo
                {
                        Assert::IsTrue(sub(2, 2) == 0); // Verifica se a subtração de 2 - 2 é igual a 0
                }
        };

        TEST_CLASS(Multipicação) // Classe de testes para a função de multiplicação
        {
        public:

                TEST_METHOD(Zero)                   // Método de teste para multiplicações com valor zero
                {
                        Assert::IsTrue(mult(0, 0) == 0); // Verifica se a multiplicação de 0 * 0 é igual a 0
                        Assert::IsTrue(mult(0, 5) == 0); // Verifica se a multiplicação de 0 * 5 é igual a 0
                }
                TEST_METHOD(Básico)                  // Método de teste para multiplicações básicas
                {
                        Assert::IsTrue(mult(3, 4) == 12);   // Verifica se a multiplicação de 3 * 4 é igual a 12
                        Assert::IsTrue(mult(-2, 5) == -10); // Verifica se a multiplicação de -2 * 5 é igual a
//-10
                }
                TEST_METHOD(Comutativa)             // Método de teste para a propriedade comutativa da
//multiplicação
```

```
                {
                        Assert::AreEqual(mult(2, 5), mult(5, 2)); // Verifica se a multiplicação de 2 * 5 é
//igual à multiplicação de 5 * 2
                }
                TEST_METHOD(False)              // Método de teste para verificar falso
                {
                        Assert::IsFalse(mult(0, 0) == 1); // Verifica se a multiplicação de 0 * 0 não é igual a
//1
                }
        };
        TEST_CLASS(Divisão) // Classe de testes para a função de divisão
        {
        public:
                TEST_METHOD(divi)               // Método de teste da divisão básica
                {
                        Assert::AreEqual(divide(10, 5), 2); // Compara o resultado da divisão com o valor
//esperado
                }
                TEST_METHOD(zero)               // Método de teste para a divisão por zero
                {
                        // Tenta chamar a função de divisão com um segundo argumento igual a zero
                        try {

                                divide(10, 0);          // Chama a função de divisão com um segundo
//argumento igual a zero

                                Assert::Fail();         // Se a função não lançar uma exceção, o teste falha
                        }
                        // Se a função lançar uma exceção, verifica se a mensagem da exceção é a
//esperada

                        catch (std::invalid_argument const& e) {
                                Assert::AreEqual(e.what(), "Divisão por zero.");
                        }
                }
        };
}
```

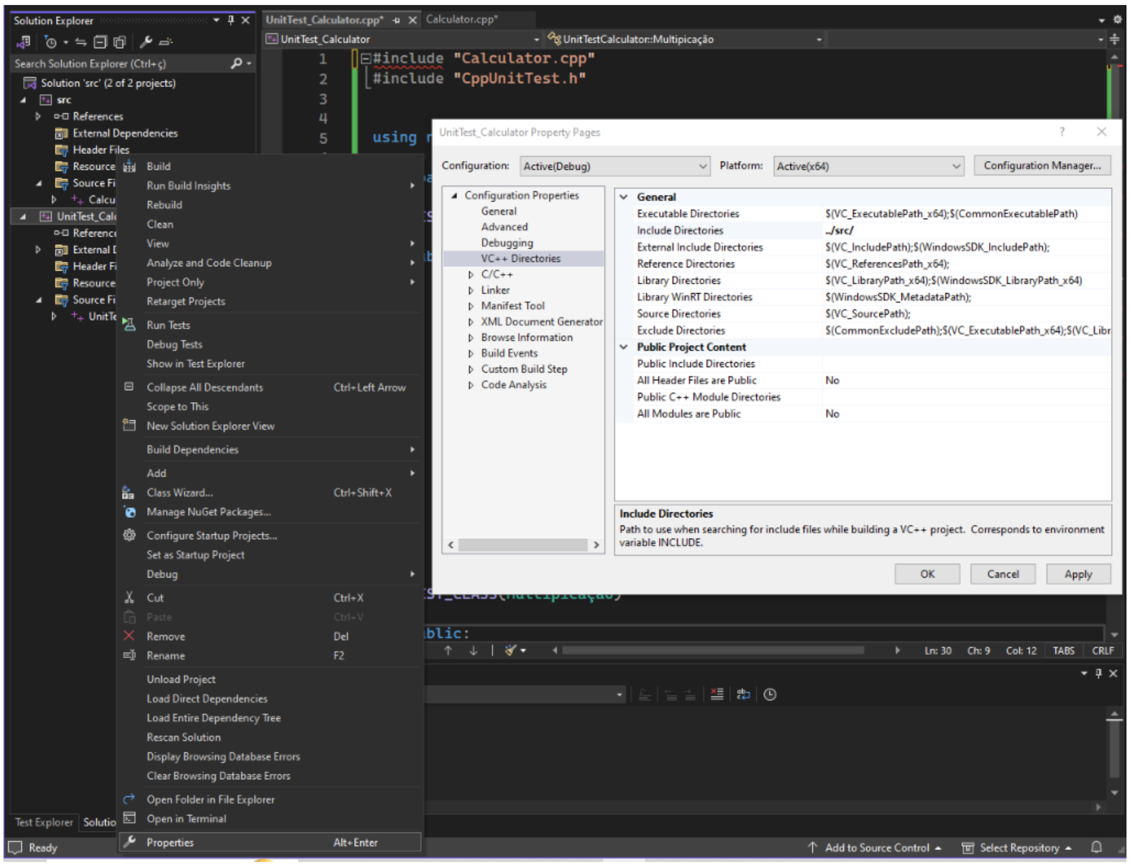


```
#include "Calculator.cpp"
#include "CppUnitTest.h"


using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestCalculator
{
    TEST_CLASS(Soma)
    {
    public:

        TEST_METHOD(Zero)
        {
            Assert::IsTrue(soma(0, 0) == 0);
```

It's evident that when you insert the code, the program can't find the library where the required functions are located to execute the tests. This is because it's unable to locate the functions that are in another project, in another file. To make it recognize them, we need to provide the path so that the software can find the file with the functions.

This can be done by right-clicking on the test cases project folder, in this case, the "UnitTest_Calculator" folder -> "Properties" -> "Configuration Properties" -> "VC++ Directories," and then add the "<Directory of the folder where the library you want to test is

located>". If you're following this tutorial, simply navigate back one directory and enter the "src" directory.



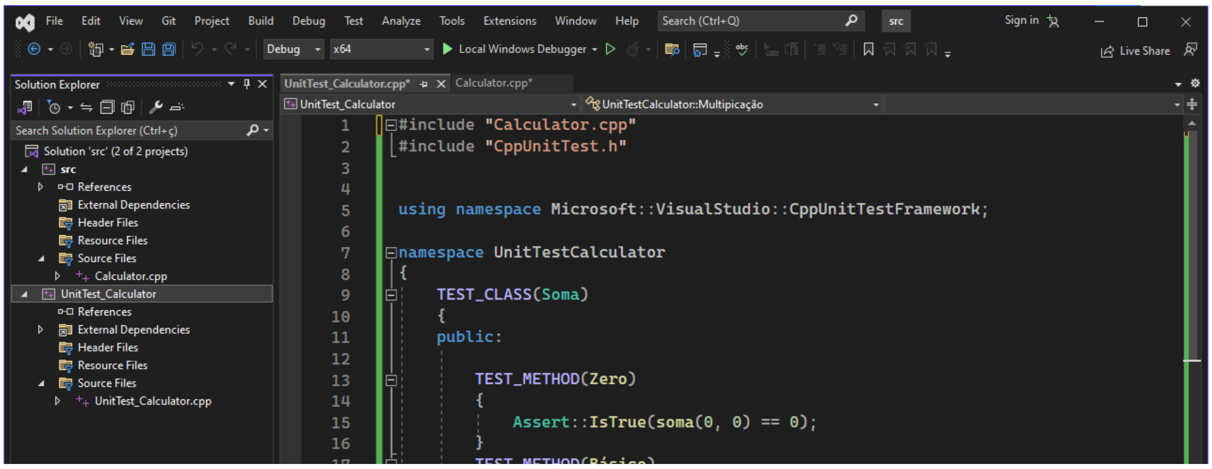
```
#include "Calculator.cpp"
#include "CppUnitTest.h"


using
```

Configuration: Active(Debug) Platform: Active(x64)

General	
Executable Directories	$(VC_ExecutablePath_x64);$(CommonExecutablePath)
Include Directories	../src/
External Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
Reference Directories	$(VC_ReferencesPath_x64);
Library Directories	$(VC_LibraryPath_x64);$(WindowsSDK_LibraryPath_x64)
Library WinRT Directories	$(WindowsSDK_MetadataPath);
Source Directories	$(VC_SourcePath);
Exclude Directories	$(CommonExcludePath);$(VC_ExecutablePath_x64);$(VC_Libr
Public Project Content	
Public Include Directories	
All Header Files are Public	No
Public C++ Module Directories	
All Modules are Public	No

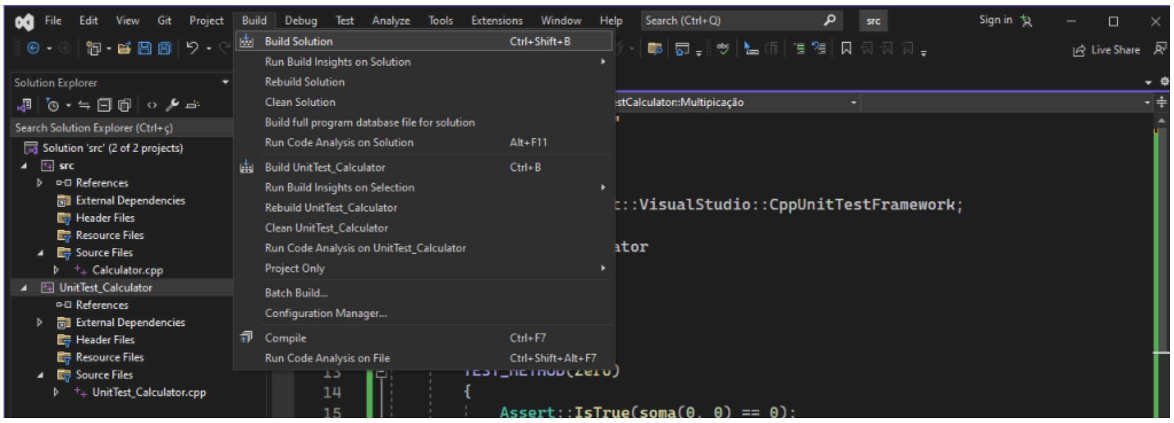
Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

As you can see in the image below, after performing the procedure, the error mark disappears from the code, and now you can run the tests.

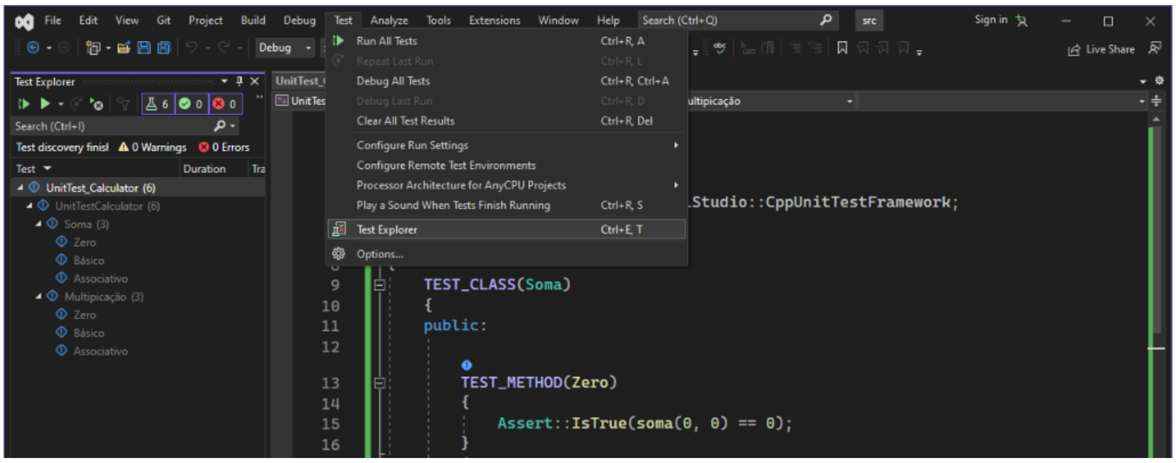


```
#include "Calculator.cpp"
#include "CppUnitTest.h"


using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestCalculator
{
    TEST_CLASS(Soma)
    {
    public:

        TEST_METHOD(Zero)
        {
            Assert::IsTrue(soma(0, 0) == 0);
        }
        TEST_METHOD(Básico)
```

Step 6 - To compile the files, you can click on the "Build" option and then select "Build Solution."


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
src
Sign in
Live Share
Build Solution Ctrl+Shift+B
Run Build Insights on Solution
Rebuild Solution
Clean Solution
Build full program database file for solution
Run Code Analysis on Solution Alt+F11
Build UnitTest_Calculator Ctrl+B
Run Build Insights on Selection
Rebuild UnitTest_Calculator
Clean UnitTest_Calculator
Run Code Analysis on UnitTest_Calculator
Project Only
Batch Build...
Configuration Manager...
Compile Ctrl+F7
Run Code Analysis on File Ctrl+Shift+Alt+F7
Solution Explorer
Search Solution Explorer (Ctrl+ç)
Solution 'src' (2 of 2 projects)
src
References
External Dependencies
Header Files
Resource Files
Source Files
Calculator.cpp
UnitTest_Calculator
References
External Dependencies
Header Files
Resource Files
Source Files
UnitTest_Calculator.cpp
stCalculator::Multipicação

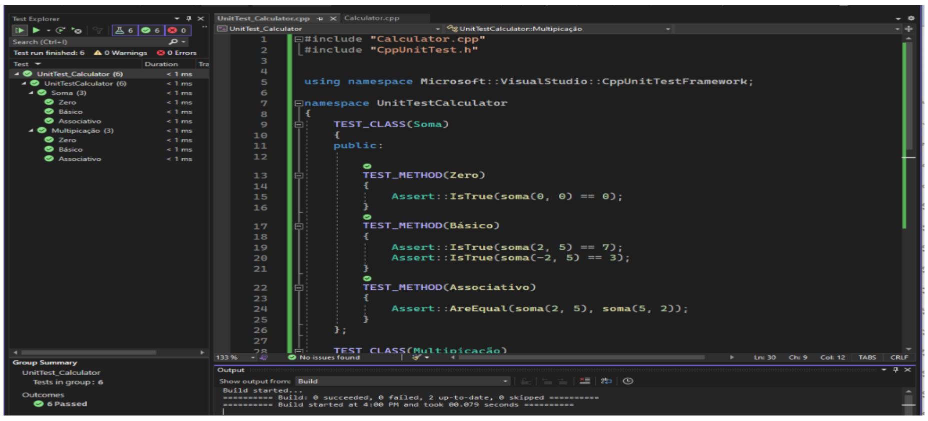
To analyze the tests, you can open the "Test Explorer," which can be found in the top menu by clicking on the "Build" tab -> "Test Explorer."


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
src
Sign in
Live Share
Debug
Run All Tests Ctrl+R, A
Repeat Last Run Ctrl+R, L
Debug All Tests Ctrl+R, Ctrl+A
Debug Last Run Ctrl+R, D
Clear All Test Results Ctrl+R, Del
Configure Run Settings
Configure Remote Test Environments
Processor Architecture for AnyCPU Projects
Play a Sound When Tests Finish Running Ctrl+R, S
Test Explorer Ctrl+E, T
Options...
Test Explorer
6 0 0
Search (Ctrl+I)
Test discovery finisl 0 Warnings 0 Errors
Test Duration Tra
UnitTest_Calculator (6)
UnitTestCalculator (6)
Soma (3)
Zero
Básico
Associativo
Multipicação (3)
Zero
Básico
Associativo
ultipicação

```
    TEST_CLASS(Soma)
    {
    public:

        TEST_METHOD(Zero)
        {
            Assert::IsTrue(soma(0, 0) == 0);
        }
```

Now, with the "Test Explorer" open, when you click the "Run" button, the tests will be executed and displayed. When a test "passes," it turns green, and when a test "fails," it turns red.

In the image below, you can see the results for the tests that were mentioned earlier.



```
#include "Calculator.cpp"
#include "CppUnitTest.h"


using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestCalculator
{
    TEST_CLASS(Soma)
    {
    public:

        TEST_METHOD(Zero)
        {
            Assert::IsTrue(soma(0, 0) == 0);
        }
        TEST_METHOD(Básico)
        {
            Assert::IsTrue(soma(2, 5) == 7);
            Assert::IsTrue(soma(-2, 5) == 3);
        }
        TEST_METHOD(Associativo)
        {
            Assert::AreEqual(soma(2, 5), soma(5, 2));
        }
    };

    TEST_CLASS(Multipicação)
```

```
Build started...
========== Build: 0 succeeded, 0 failed, 2 up-to-date, 0 skipped ==========
========== Build started at 4:00 PM and took 00.079 seconds ==========
```

Methods Used for this Tutorial:

Assert::IsTrue()	Checks if an expression is true.
Assert::IsFalse()	Checks if an expression is false.
Assert::AreEqual()	Checks if two values are equal.
Assert::Fail()	Fails the current test.