You said: create a c++ file math_operations.h with declaration
create a c++ file math_operations.h with declaration of function int add(int a, int b)

```cpp name=math_operations.h
#pragma once

// Declaration of a simple addition function
int add(int a, int b);
```

I created math_operations.h with a clean include guard (pragma once) and the declaration int add(int a, int b). Next I can add an implementation (math_operations.cpp) and update your CMakeLists.txt to compile and link it into the HelloWorld executable if you'd like.

implement the function int add(int a, int b) that includes math_operations.h and returns the sum of a and b

```cpp name=math_operations.cpp
#include "math_operations.h"

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

I implemented the add function in math_operations.cpp which includes math_operations.h and returns the sum of a and b. Next you can add this file to your build (for example, add math_operations.cpp to your CMakeLists.txt) and call add(...) from main.cpp to verify it works.

generate a c++ file units_tests.cpp with Google test unit tests for the function int add(int a, int b) from math_operations.h in tests/unit_tests.cpp. Include tests for positive numbers, zero and negative numbers

```cpp name=tests/unit_tests.cpp
#include <gtest/gtest.h>
#include "math_operations.h"

// Tests for add with positive numbers
TEST(AdditionPositiveTests, SmallNumbers) {
    EXPECT_EQ(add(2, 3), 5);
    EXPECT_EQ(add(1, 1), 2);
}

TEST(AdditionPositiveTests, LargerNumbers) {
    EXPECT_EQ(add(100, 200), 300);
    EXPECT_EQ(add(12345, 67890), 80235);
}

// Tests involving zero
TEST(AdditionZeroTests, ZeroAndZero) {
    EXPECT_EQ(add(0, 0), 0);
}

TEST(AdditionZeroTests, ZeroAndNonZero) {
    EXPECT_EQ(add(0, 5), 5);
    EXPECT_EQ(add(-5, 0), -5);
}

// Tests with negative numbers
TEST(AdditionNegativeTests, BothNegative) {
    EXPECT_EQ(add(-2, -3), -5);
    EXPECT_EQ(add(-100, -200), -300);
}

TEST(AdditionNegativeTests, MixedSigns) {
    EXPECT_EQ(add(-10, 4), -6);
    EXPECT_EQ(add(7, -3), 4);
}

// Optional: a main so this test file can be built standalone (if you don't link gtest_main)
int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

I created tests/unit_tests.cpp with Google Test cases covering positive numbers, zero, negative numbers, and mixed-sign cases. The file includes a main() so it can be run standalone (link against GoogleTest). To build and run these tests, add this file to your test target and link with gtest (or gtest_main) in your CMakeLists.txt or build system.

now add Google test support to my existing CMakeLists.txt file

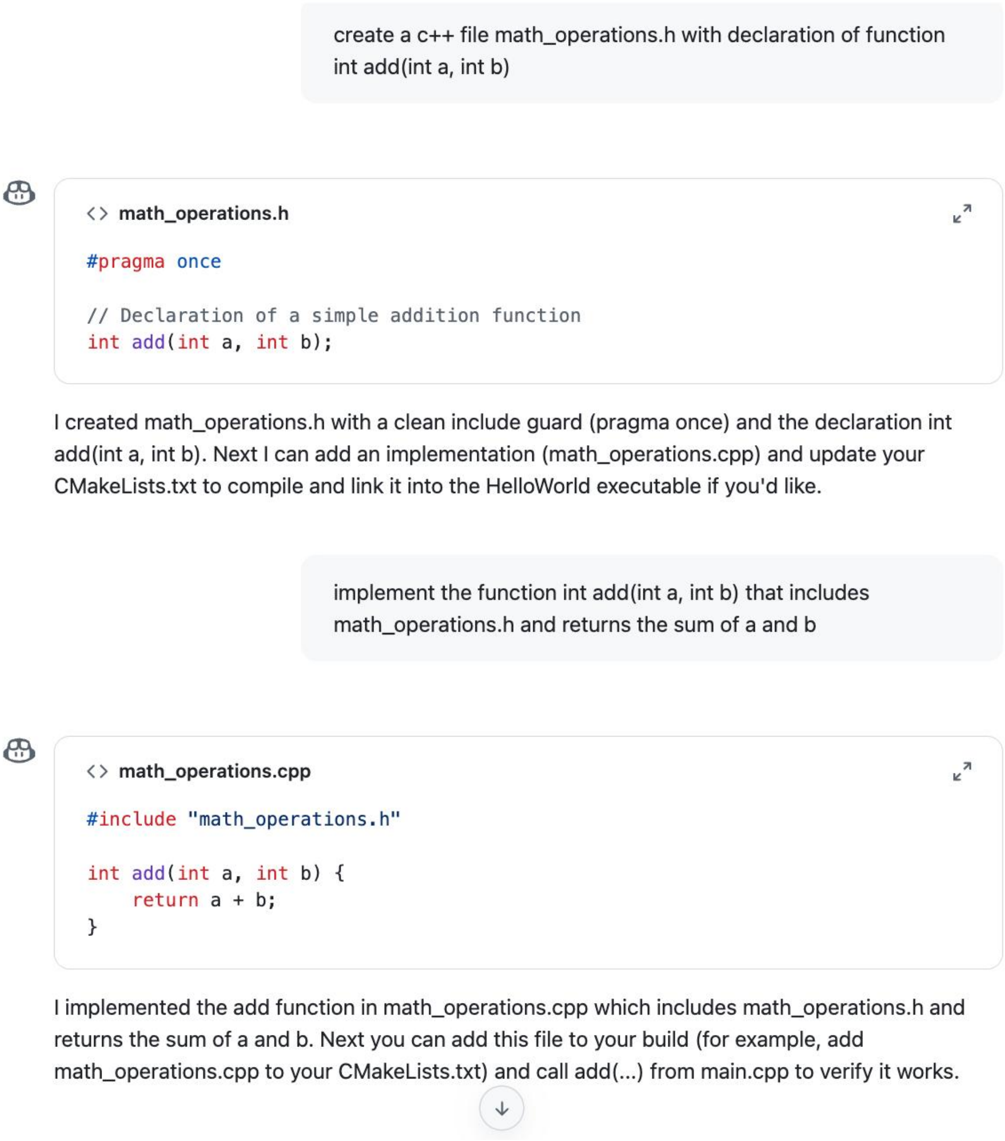
```name=CMakeLists.txt
cmake_minimum_required(VERSION 3.14)

project(HelloWorld VERSION 1.0 LANGUAGES CXX)

# Require C++17
```

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)

# Create a small library for math operations so both the executable and tests can link to it
add_library(math_ops STATIC
  math_operations.cpp
)

target_include_directories(math_ops PUBLIC
  $<BUILD_INTERFACE:${CMAKE_SOURCE_DIR}>
)

# Create the main executable from main.cpp and link the math_ops library
add_executable(HelloWorld main.cpp)
target_link_libraries(HelloWorld PRIVATE math_ops)

# Prefer target-based properties
target_compile_features(HelloWorld PRIVATE cxx_std_17)
target_compile_features(math_ops PRIVATE cxx_std_17)

# Warning level / pedantic flags per compiler (helpful cross-platform defaults)
if (MSVC)
  target_compile_options(HelloWorld PRIVATE /W4 /permissive-)
  target_compile_options(math_ops PRIVATE /W4 /permissive-)
else ()
  target_compile_options(HelloWorld PRIVATE -Wall -Wextra -Wpedantic)
  target_compile_options(math_ops PRIVATE -Wall -Wextra -Wpedantic)
endif()

# Place binaries in a predictable folder inside the build directory
set_target_properties(HelloWorld PROPERTIES
  RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/bin"
)

# Install rule for the produced executable
install(TARGETS HelloWorld
  RUNTIME DESTINATION bin
)

# Option to enable building tests (OFF by default)
option(BUILD_TESTS "Build tests" OFF)

# GoogleTest integration (download & build when BUILD_TESTS is ON)
if (BUILD_TESTS)
  enable_testing()
  include(FetchContent)

  # Fetch GoogleTest; this will download the project at configure time if needed.
  FetchContent_Declare(
    googletest
    URL https://github.com/google/googletest/archive/refs/heads/main.zip
  )

  # For MSVC: prefer shared CRT for gtest to match project settings if needed
  if (MSVC)
    set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
  endif()
  FetchContent_MakeAvailable(googletest)

  # Add the test executable and link to gtest_main and our math_ops library
  add_executable(unit_tests tests/unit_tests.cpp)
  target_link_libraries(unit_tests PRIVATE GTest::gtest_main math_ops)
  target_include_directories(unit_tests PRIVATE ${CMAKE_SOURCE_DIR})
  target_compile_features(unit_tests PRIVATE cxx_std_17)

  if (MSVC)
    target_compile_options(unit_tests PRIVATE /W4 /permissive-)
  else ()
    target_compile_options(unit_tests PRIVATE -Wall -Wextra -Wpedantic)
  endif()

  # Put tests in the same predictable output folder
  set_target_properties(unit_tests PROPERTIES
    RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/bin"
  )

  add_test(NAME unit_tests COMMAND unit_tests)
endif()
```

I updated your CMakeLists.txt to:
- Build a small static library (math_ops) from math_operations.cpp and link it into the HelloWorld executable.
- Add an option BUILD_TESTS (default OFF) that, when enabled, downloads GoogleTest via FetchContent, builds a unit_tests executable from tests/unit_tests.cpp, links it with GTest and math_ops, and registers it with CTest.

How to use:
- To build normally: mkdir build && cd build && cmake .. && cmake --build .
- To build tests: mkdir build && cd build && cmake -DBUILD_TESTS=ON .. && cmake --build . && ctest --output-on-failure

Tell me if you want GoogleTest to be assumed installed on the system (find_package) instead of fetched, or if you prefer a different minimum CMake version or a different test discovery setup.



create a c++ file math_operations.h with declaration of function int add(int a, int b)

math_operations.h

```
#pragma once

// Declaration of a simple addition function
int add(int a, int b);
```

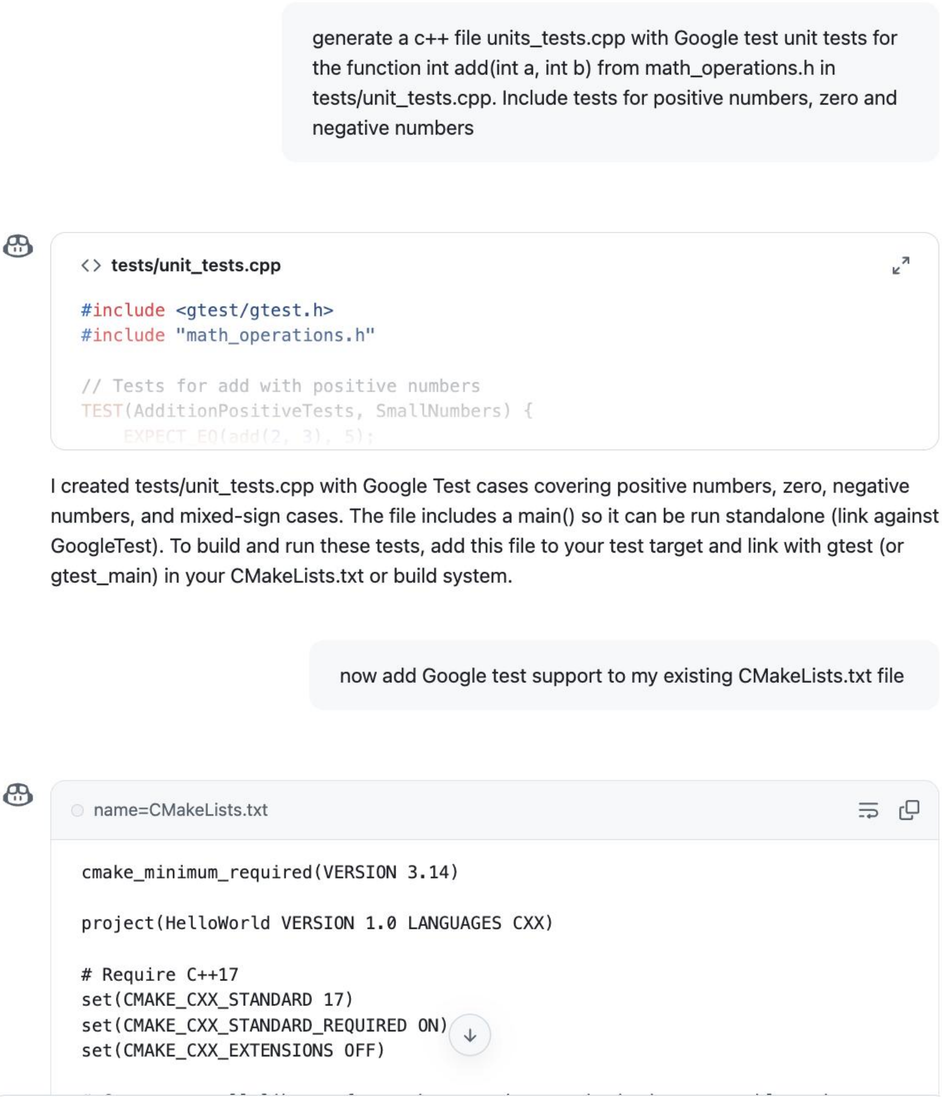
I created math_operations.h with a clean include guard (pragma once) and the declaration int add(int a, int b). Next I can add an implementation (math_operations.cpp) and update your CMakeLists.txt to compile and link it into the HelloWorld executable if you'd like.

implement the function int add(int a, int b) that includes math_operations.h and returns the sum of a and b

math_operations.cpp

```
#include "math_operations.h"

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

I implemented the add function in math_operations.cpp which includes math_operations.h and returns the sum of a and b. Next you can add this file to your build (for example, add math_operations.cpp to your CMakeLists.txt) and call add(...) from main.cpp to verify it works.



generate a c++ file units_tests.cpp with Google test unit tests for the function int add(int a, int b) from math_operations.h in tests/unit_tests.cpp. Include tests for positive numbers, zero and negative numbers

tests/unit_tests.cpp

```cpp
#include <gtest/gtest.h>
#include "math_operations.h"

// Tests for add with positive numbers
TEST(AdditionPositiveTests, SmallNumbers) {
    EXPECT_EQ(add(2, 3), 5);
```

I created tests/unit_tests.cpp with Google Test cases covering positive numbers, zero, negative numbers, and mixed-sign cases. The file includes a main() so it can be run standalone (link against GoogleTest). To build and run these tests, add this file to your test target and link with gtest (or gtest_main) in your CMakeLists.txt or build system.

now add Google test support to my existing CMakeLists.txt file

name=CMakeLists.txt

```cmake
cmake_minimum_required(VERSION 3.14)

project(HelloWorld VERSION 1.0 LANGUAGES CXX)

# Require C++17
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)
```

```
  # Add the test executable and link to gtest_main and our math_ops library
  add_executable(unit_tests tests/unit_tests.cpp)
  target_link_libraries(unit_tests PRIVATE GTest::gtest_main math_ops)
  target_include_directories(unit_tests PRIVATE ${CMAKE_SOURCE_DIR})
  target_compile_features(unit_tests PRIVATE cxx_std_17)

  if (MSVC)
    target_compile_options(unit_tests PRIVATE /W4 /permissive-)
  else ()
    target_compile_options(unit_tests PRIVATE -Wall -Wextra -Wpedantic)
  endif()

  # Put tests in the same predictable output folder
  set_target_properties(unit_tests PROPERTIES
    RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/bin"
  )

  add_test(NAME unit_tests COMMAND unit_tests)
endif()
```

I updated your CMakeLists.txt to:

- Build a small static library (math_ops) from math_operations.cpp and link it into the HelloWorld executable.
- Add an option BUILD_TESTS (default OFF) that, when enabled, downloads GoogleTest via FetchContent, builds a unit_tests executable from tests/unit_tests.cpp, links it with GTest and math_ops, and registers it with CTest.

How to use:

- To build normally: mkdir build && cd build && cmake .. && cmake --build .
- To build tests: mkdir build && cd build && cmake -DBUILD_TESTS=ON .. && cmake --build . && ctest --output-on-failure

Tell me if you want GoogleTest to be assumed installed on the system (find_package) instead of fetched, or if you prefer a different minimum CMake version or a different test discovery setup.

Ask anything

+ Add repositories, files, and spaces

GPT-5 mini