add CMake configuration for enabling testing and running unit tests using CTest

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

project(HelloWorld VERSION 1.0 LANGUAGES CXX)

# Provide a top-level CTest integration so we can create a 'check' target that runs tests.
# include(CTest) exposes CTest-related variables and the CTEST_TESTER environment variable.
include(CTest)

# 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 and register CTest 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"
)

# Register the test with CTest so it can be run with 'ctest'
add_test(NAME unit_tests COMMAND unit_tests)

# Provide a convenient 'check' target that runs CTest with verbose output on failure.
# Users can run: cmake --build . --target check
add_custom_target(check
  COMMAND ${CMAKE_CTEST_COMMAND} --output-on-failure
  DEPENDS unit_tests
  WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
  )
endif()
```

I updated your CMakeLists.txt to:

- Include CTest integration at the top (include(CTest)).

- Call enable_testing() when BUILD_TESTS is ON and register the unit_tests executable with add_test.

- Add a convenient custom target named check which runs CTest (so you can run tests via: cmake --build . --target check).

- Kept the existing FetchContent-based GoogleTest integration and the math_ops library so both your app and tests share the same code.

How to build and run tests:

- Configure with tests enabled:

  - mkdir build && cd build

  - cmake -DBUILD_TESTS=ON ..

- Build everything and run tests:

  - cmake --build .   # builds hello and unit_tests

  - cmake --build . --target check   # runs ctest with output on failure

  - or run: ctest --output-on-failure

If you'd like BUILD_TESTS to default to ON, or prefer to use a system-installed GoogleTest via find_package(GTest REQUIRED) instead of FetchContent, tell me and I'll update the file.

Generate a Windows batch script to build and test a CMake project. The script should create a build directorym go inside it, run cmake .., build the project, and run ctes

```name=build_and_test.bat
@echo off
REM build_and_test.bat
REM Creates a build directory, runs CMake configure, builds the project, and runs CTest.
REM Usage: build_and_test.bat [Configuration]
REM Example: build_and_test.bat Release
REM If no configuration is provided, Release is used by default.

setlocal

:: Allow user to provide configuration (Release, Debug, etc.)
set "CONFIG=%~1"
if "%CONFIG%"=="" set "CONFIG=Release"

echo.
echo ===== Build and Test (CMake) =====
echo Configuration: %CONFIG%
echo ==================================
echo.

:: Create build directory if it doesn't exist
if not exist "build" (
  echo Creating build directory...
  mkdir "build"
  if errorlevel 1 (
    echo Failed to create build directory.
    endlocal
    exit /b 1
  )
)

pushd "build" || (
  echo Failed to enter build directory.
  endlocal
  exit /b 1
)

echo Running CMake configure...
REM For single-config generators this sets CMAKE_BUILD_TYPE; for multi-config it's ignored.
```

```
cmake .. -DCMAKE_BUILD_TYPE=%CONFIG%
if errorlevel 1 (
  echo CMake configuration failed.
  popd
  endlocal
  exit /b 1
)

echo Building project...
REM Use CMake's --parallel for parallel build if supported; falls back if not supported by older CMake.
set "JOBS=%NUMBER_OF_PROCESSORS%"
if "%JOBS%"=="" set "JOBS=2"

cmake --build . --config %CONFIG% --parallel %JOBS%
if errorlevel 1 (
  echo Build failed.
  popd
  endlocal
  exit /b 1
)

echo Running tests with CTest...
ctest --output-on-failure -C %CONFIG%
if errorlevel 1 (
  echo Some tests failed.
  popd
  endlocal
  exit /b 1
)

echo.
echo All steps completed successfully.
popd
endlocal
```

```
exit /b 0
```

I created build_and_test.bat which:

- Creates a build directory (if needed) and changes into it.

- Runs CMake configure (cmake ..), passing CMAKE_BUILD_TYPE for single-config generators.

- Builds the project using cmake --build with parallel jobs based on the NUMBER_OF_PROCESSORS environment variable.

- Runs tests with ctest --output-on-failure for the provided configuration (default: Release).

How to use: open a Developer Command Prompt or regular Command Prompt with CMake on PATH, then run:

- build_and_test.bat

or

- build_and_test.bat Debug

If you want the script to force a particular generator, clean the build directory before configuring, or produce more detailed logs, tell me and I'll update the script.

add CMake configuration for enabling testing and running unit tests using CTest

name=CMakeLists.txt

```
cmake_minimum_required(VERSION 3.14)

project(HelloWorld VERSION 1.0 LANGUAGES CXX)

# Provide a top-level CTest integration so we can create a 'check' target that runs tests
# include(CTest) exposes CTest-related variables and the CTEST_TESTER environment variabl
include(CTest)

# 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
add_library(math_ops STATIC
  math_operations.cpp
)
```

Generate a Windows batch script to build and test a CMake project. The script should create a build directorym go inside it, run cmake .., build the project, and run ctes

name=build_and_test.bat

```
@echo off
REM build_and_test.bat
REM Creates a build directory, runs CMake configure, builds the project, and runs CTest.
REM Usage: build_and_test.bat [Configuration]
REM Example: build_and_test.bat Release
REM If no configuration is provided, Release is used by default.

setlocal

:: Allow user to provide configuration (Release, Debug, etc.)
set "CONFIG=%~1"
if "%CONFIG%"=="" set "CONFIG=Release"

echo.
echo ===== Build and Test (CMake) =====
echo Configuration: %CONFIG%
echo =================================
echo.
```