STEPS

CMake File (I've mentioned the changes as comments corresponding to the line which ive added manually)

```
cmake_minimum_required(VERSION 3.14)

# Set project name, version and laguages here. (change as needed)
# Version numbers are available by including "exampleConfig.h" in
# the source. See exampleConfig.h.in for some more details.
```
project(CPP_BOILERPLATE VERSION 1.2.3.4 LANGUAGES C CXX)
```
## Manually added C alongside CXX to work with C also


# Options: Things you can set via commandline options to cmake (e.g.
-DENABLE_LTO=[ON|OFF])
option(ENABLE_WARNINGS_SETTINGS "Allow target_set_warnings to add flags and defines.
                        Set this to OFF if you want to provide your own warning parameters." ON)
option(ENABLE_LTO "Enable link time optimization" ON)
option(ENABLE_DOCTESTS "Include tests in the library. Setting this to OFF will remove all
doctest related code.
                  Tests in tests/*.cpp will still be enabled." ON)

# Include stuff. No change needed.
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH}
"${CMAKE_SOURCE_DIR}/cmake/")
include(ConfigSafeGuards)
include(Colors)
include(CTest)
include(Doctest)
include(Documentation)
include(LTO)
include(Misc)
include(Warnings)

# Check for LTO support.
find_lto(CXX)

# ---------------------------------------------------------------------------
#                    Locate files (change as needed).
# ---------------------------------------------------------------------------
set(SOURCES         # All .cpp files in src/
   src/example.cpp
   **src/hello.c          # Manually added this file now for testing if C is working**
)
set(TESTFILES       # All .cpp files in tests/
   tests/main.cpp
)
set(LIBRARY_NAME engine)  # Default name for the library built from src/*.cpp (change if you
wish)

# ---------------------------------------------------------------------------
#                        Build! (Change as needed)
# ---------------------------------------------------------------------------
# Compile all sources into a library.
add_library(${LIBRARY_NAME} OBJECT ${SOURCES})

# Lib needs its header files, and users of the library must also see these (PUBLIC). (No change
needed)
target_include_directories(${LIBRARY_NAME} PUBLIC ${PROJECT_SOURCE_DIR}/include)

# There's also (probably) doctests within the library, so we need to see this as well.
target_link_libraries(${LIBRARY_NAME} PUBLIC doctest)

# Set the compile options you want (change as needed).
target_set_warnings(${LIBRARY_NAME} ENABLE ALL AS_ERROR ALL DISABLE Annoying)
# target_compile_options(${LIBRARY_NAME} ... )  # For setting manually.

# Add an executable for the file app/main.cpp.
# If you add more executables, copy these lines accordingly.
add_executable(main app/main.cpp)   # Name of exec. and location of file.
target_link_libraries(main PRIVATE ${LIBRARY_NAME})  # Link the executable to library (if it
uses it).
target_set_warnings(main ENABLE ALL AS_ERROR ALL DISABLE Annoying) # Set warnings
(if needed).
target_enable_lto(main optimized)

## code for testing GTest

# enable link-time-optimization if available for non-debug configurations

# Set the properties you require, e.g. what C++ standard to use. Here applied to library and
main (change as needed).
set_target_properties(
   ${LIBRARY_NAME} main
     PROPERTIES
       CXX_STANDARD 17
```

```
    CXX_STANDARD_REQUIRED YES
    CXX_EXTENSIONS NO
)

# Set up tests (see tests/CMakeLists.txt).
```

```
# Below block is completeley added manually to test gtest framework so what this does is it only uses the gtest within the package the cmake refers itself and will work only within this package if needed can download gtest locally in system if you want
include(FetchContent)

FetchContent_Declare(
  googletest
  URL https://github.com/google/googletest/archive/refs/heads/main.zip
)
# For Windows: Prevent overriding the parent project's compiler/linker settings
set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
FetchContent_MakeAvailable(googletest)

add_executable(gtest_test tests/gtest_test.cpp)      # Manually created a new file called gtest_test.cpp to test if gtest framework is working.
target_link_libraries(gtest_test PRIVATE gtest_main)
add_test(NAME GTestSmokeTest COMMAND gtest_test)
##### Manuallly added code block ends here before add_subdirectory(tests)############################################
```

```
add_subdirectory(tests)
```

GTEST CODE

Under tests folder in cpp-project create a new file called gtest_test.cpp

cpp-project→tests→ gtest_test.cpp

Gtest_test.cpp

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

TEST(SmokeTest, AlwaysPasses) {
   EXPECT_EQ(1, 1);
}
```

Now go to build directory under cpp-project to build and run the gtest code

```
cmake ..
make
ctest
```

Checking if C is working

Create a file hello.h under cpp-project→include → hello.h

hello.h

```
#ifndef HELLO_H
#define HELLO_H

void say_hello();

#endif
```

Now create a new file hello.c under cpp-project→src→hello.c

hello.c

```
#include <stdio.h>

void say_hello() {
   printf("Hello from C!\n");
}
```

Now make changes in main.cpp to execute the c code from there under
cpp-project → app → main.cpp

main.cpp

```
#ifdef ENABLE_DOCTEST_IN_LIBRARY
```

```
#define DOCTEST_CONFIG_IMPLEMENT
#include "doctest/doctest.h"
#endif
extern "C" {
  #include "hello.h"    // Manually added the extern block... this is added as
cpp should not confuse C functions.
  }
#include <iostream>
#include <stdlib.h>

#include "exampleConfig.h"
#include "example.h"

/*
 * Simple main program that demontrates how access
 * CMake definitions (here the version number) from source code.
 */
int main() {
  std::cout << "C++ Boiler Plate v"
            << PROJECT_VERSION_MAJOR
            << "."
            << PROJECT_VERSION_MINOR
            << "."
            << PROJECT_VERSION_PATCH
            << "."
            << PROJECT_VERSION_TWEAK
            << std::endl;
  std::system("cat ../LICENSE");
  say_hello();                    //Manually calling this function from C to print C
code output

  // Bring in the dummy class from the example source,
  // just to show that it is accessible from main.cpp.
  Dummy d = Dummy();
  return d.doSomething() ? 0 : -1;
}
```

In this code i've added extern "C" {
  #include "hello.h"   } block to specify for cpp to not confuse with c functions and I'm calling   say_hello();   function under the main() block

To execute…

Go to cd build
cmake ..
make
./main