

generate a CMakeLists.txt for a cross-platform C++ project named "HelloWorld". The project should build main.cpp, require C++ 17 standard, and produce an executable

```
``name=CMakeLists.txt

cmake_minimum_required(VERSION 3.10)


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 executable from main.cpp
add_executable(HelloWorld main.cpp)


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


# Warning level / pedantic flags per compiler (helpful cross-platform defaults)
if (MSVC)
    target_compile_options(HelloWorld PRIVATE /W4 /permissive-)
else ()
    target_compile_options(HelloWorld 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
```

```
)
```

```
# Optional: expose an option to build tests later
```

```
option(BUILD_TESTS "Build tests" OFF)
```

```
...
```

I've created a cross-platform CMakeLists.txt that defines the HelloWorld project, enforces C++17, builds main.cpp into an executable named HelloWorld, applies reasonable compiler warning flags for MSVC and other compilers, sets a consistent output directory, and adds an install target. Next you can run the usual CMake workflow (`mkdir build && cd build && cmake .. && cmake --build .`) to build the executable; tell me if you want a different executable name, a minimum CMake version change, or added support for packaging or unit tests.

generate a CMakeLists.txt for a cross-platform C++ project named "HelloWorld". The project should build main.cpp, require C++ 17 standard, and produce an executable



```
.txt
```



generate a CMakeLists.txt for a cross-platform C++ project named "HelloWorld". The project should build main.cpp, require C++ 17 standard, and produce an executable



name=CMakeLists.txt



```
cmake_minimum_required(VERSION 3.10)

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 executable from main.cpp
add_executable(HelloWorld main.cpp)

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

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

# Optional: expose an option to build tests later
option(BUILD_TESTS "Build tests" OFF)
```

I've created a cross-platform CMakeLists.txt that defines the HelloWorld project, enforces C++17, builds main.cpp into an executable named HelloWorld, applies reasonable compiler warning flags for MSVC and other compilers, sets a consistent output directory, and adds an install target. Next you can run the usual CMake workflow (`mkdir build && cd build && cmake .. && cmake --build .`) to build the executable; tell me if you want a different executable name, a minimum CMake version change, or added support for packaging or unit tests.

Ask anything

+ Add repositories, files, and spaces

GPT-5 mini ▾

