Cross-Compiling the C++ Example Project

Pieter P

The Greeter Library

For this example, we'll create a very simple library function that just takes a name and an output stream as arguments, and that prints a greeting message to this stream. It's basically a "Hello, World!" example, but as a library for demonstration purposes.

The structure of the library will be as follows:

```
greeter
├── CMakeLists.txt
├── include
│   └── greeter
│       └── greeter.hpp
├── src
│   └── greeter.cpp
└── test
    ├── CMakeLists.txt
    └── greeter.test.cpp
```

As is very common for C++ libraries, the function prototypes/declarations will be in the header file `greeter.hpp`. The implementations for these functions are in the implementation file `greeter.cpp`.
The `CMakeLists.txt` file in the `greeter` directory specifies how the library should be compiled, and where to find the headers.
Additionally, there's a `test` folder with unit tests in `greeter.test.cpp`. The `CMakeLists.txt` file in this folder specifies how to compile and link the tests executable.

```cpp
1  #pragma once
2
3  #include <iosfwd>  // std::ostream
4  #include <string>  // std::string
5
6  namespace greeter {
7
8  /**
9   * @brief   Function that greets a given person.
10  *
11  * @param   name
12  *          The name of the person to greet.
13  * @param   os
14  *          The output stream to print the greetings to.
15  */
16 void sayHello(const std::string &name, std::ostream &os);
17
18 }  // namespace greeter
```

```cpp
1  #include <greeter/greeter.hpp>
2  #include <iostream>
3
4  namespace greeter {
5
6  void sayHello(const std::string &name, std::ostream &os) {
7      os << "Hello, " << name << "!" << std::endl;
8  }
9
10 }  // namespace greeter
```

```cmake
1  add_library(greeter
2      src/greeter.cpp
3  )
4
5  target_include_directories(greeter
6      PUBLIC
7          $<INSTALL_INTERFACE:include>
8          $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
9      PRIVATE
10         src
11 )
12
13 add_subdirectory(test)
```

```cmake
1  # Add a new executable with the name "hello-world" that is compiled from the
2  # source file "hello-world.cpp".
3  add_executable(hello-world
4      hello-world.cpp
5  )
6
7  # The "hello-world" program requires the "greeter" library.
8  # The target_link_libraries command ensures that all compiler options such as
9  # include paths are set correctly, and that the executable is linked with the
10 # library as well.
11 target_link_libraries(hello-world
12     greeter
13 )
```

```cpp
#include <greeter/greeter.hpp>  // Our own custom library

#include <iostream>  // std::cout, std::cin
#include <string>    // std::getline

int main(int argc, char *argv[]) {
    std::string name;
    if (argc > 1) {         // If the user passed arguments to our program
        name = argv[1];     // The name is the first argument
    } else {                // If not, ask the user for his name
        std::cout << "Please enter your name: ";
        std::getline(std::cin, name);
    }
    greeter::sayHello(name, std::cout);  // Greet the user
}
```

```cmake
# Add a new executable with the name "hello-world" that is compiled from the
# source file "hello-world.cpp".
add_executable(hello-world
    hello-world.cpp
)

# The "hello-world" program requires the "greeter" library.
# The target_link_libraries command ensures that all compiler options such as
# include paths are set correctly, and that the executable is linked with the
# library as well.
target_link_libraries(hello-world
    greeter
)
```