



# Programmazione

Prof. Marco Bertini

[marco.bertini@unifi.it](mailto:marco.bertini@unifi.it)

<http://www.micc.unifi.it/bertini/>

---



# Building a “Hello world” with CLion

“When debugging, novices insert corrective code; experts remove defective code.”

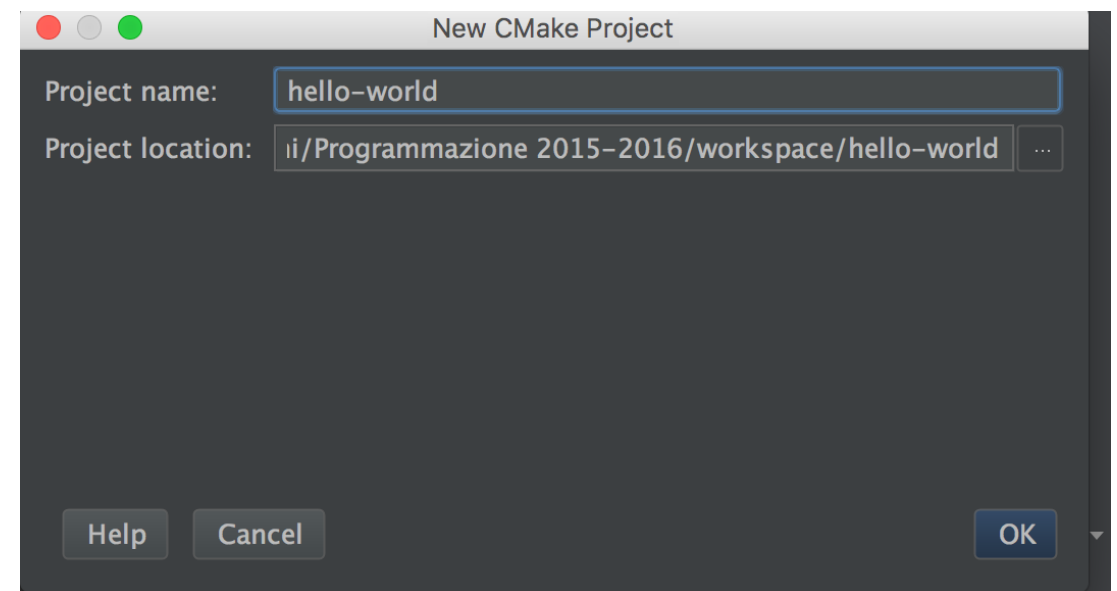
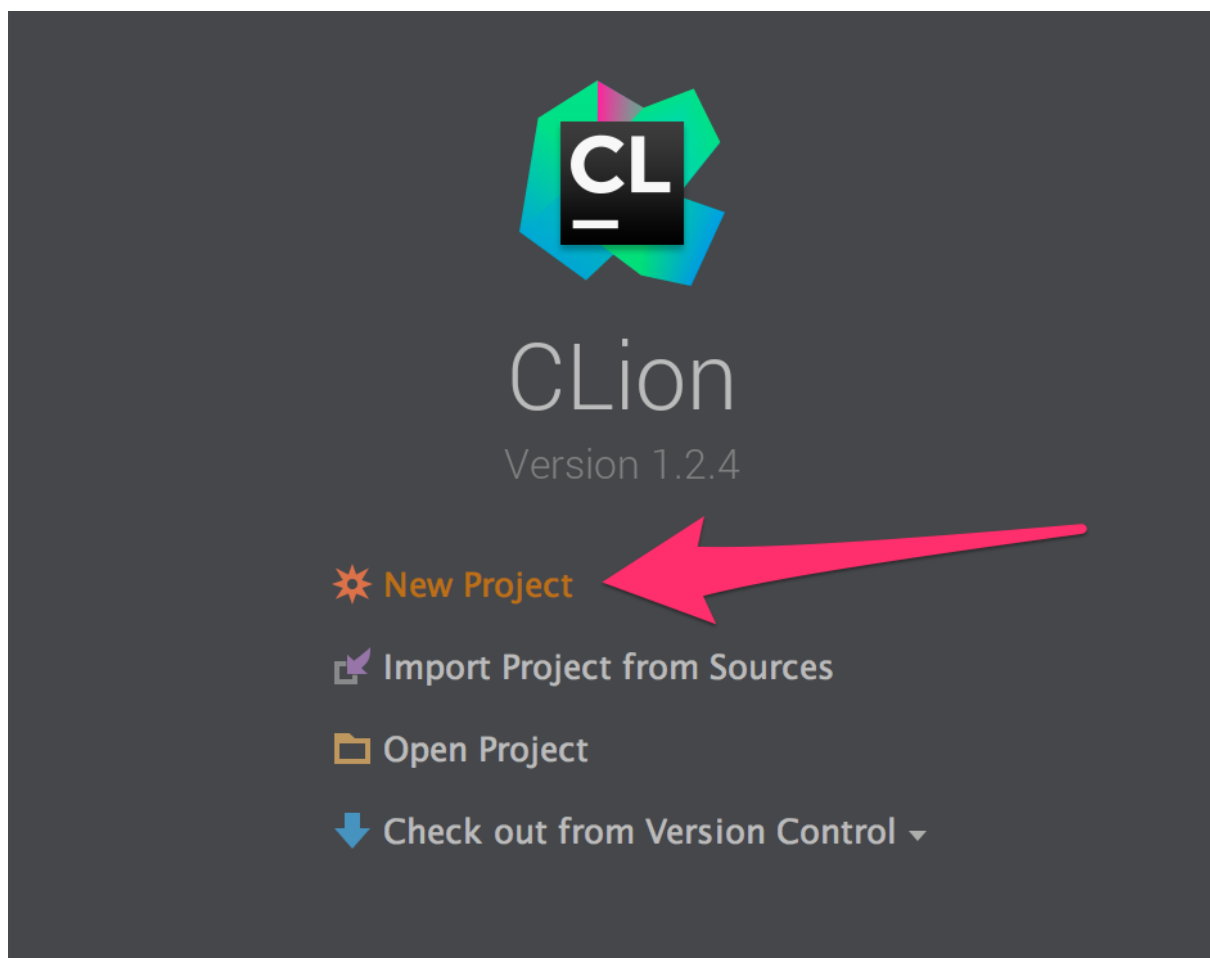
- Richard Pattis





# Use the start dialog

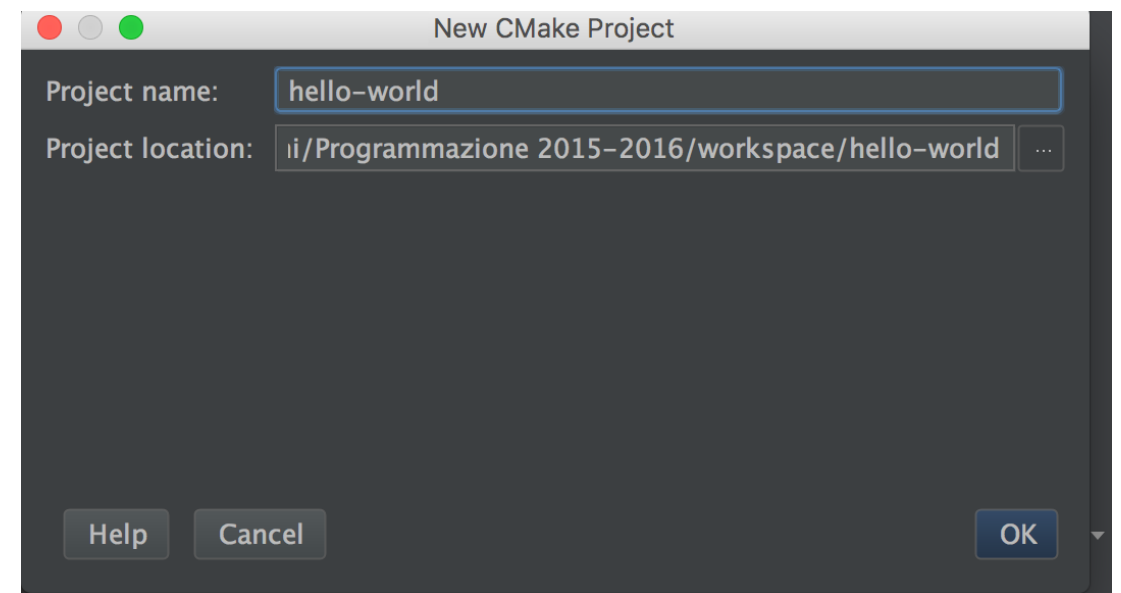
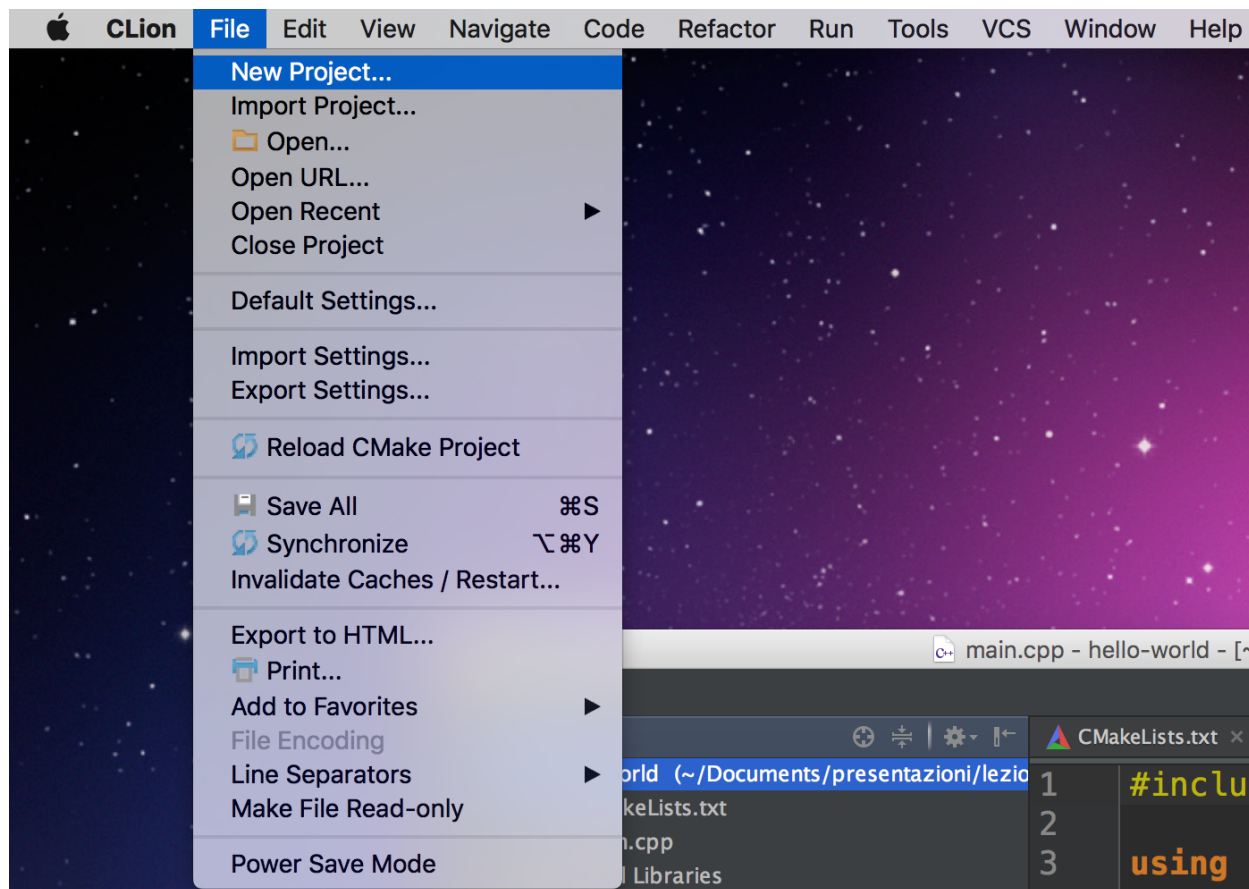
- Create a new project using the start dialog of CLion





# Use the project wizard

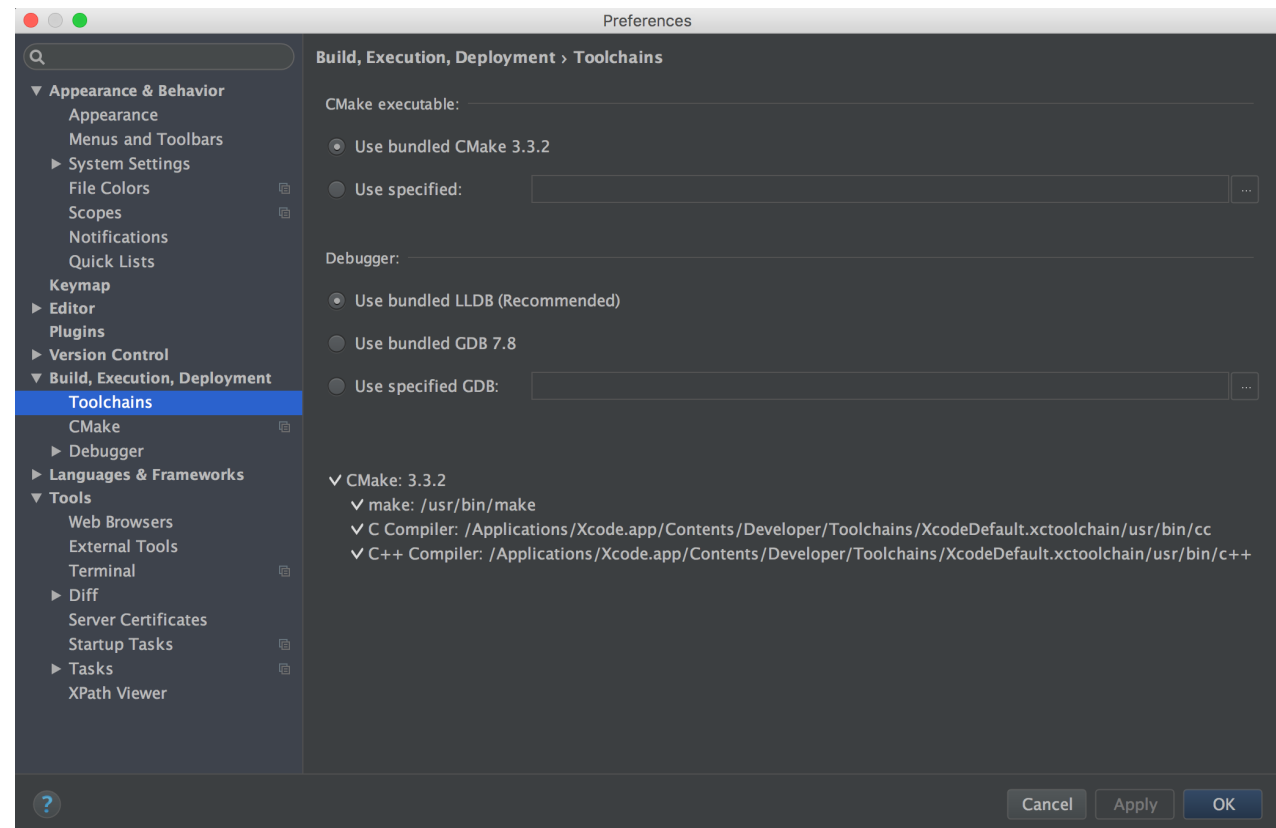
- File > New Project
- You can decide to use the old window or create a new wind, to work on two projects at the same time





# Select the tools

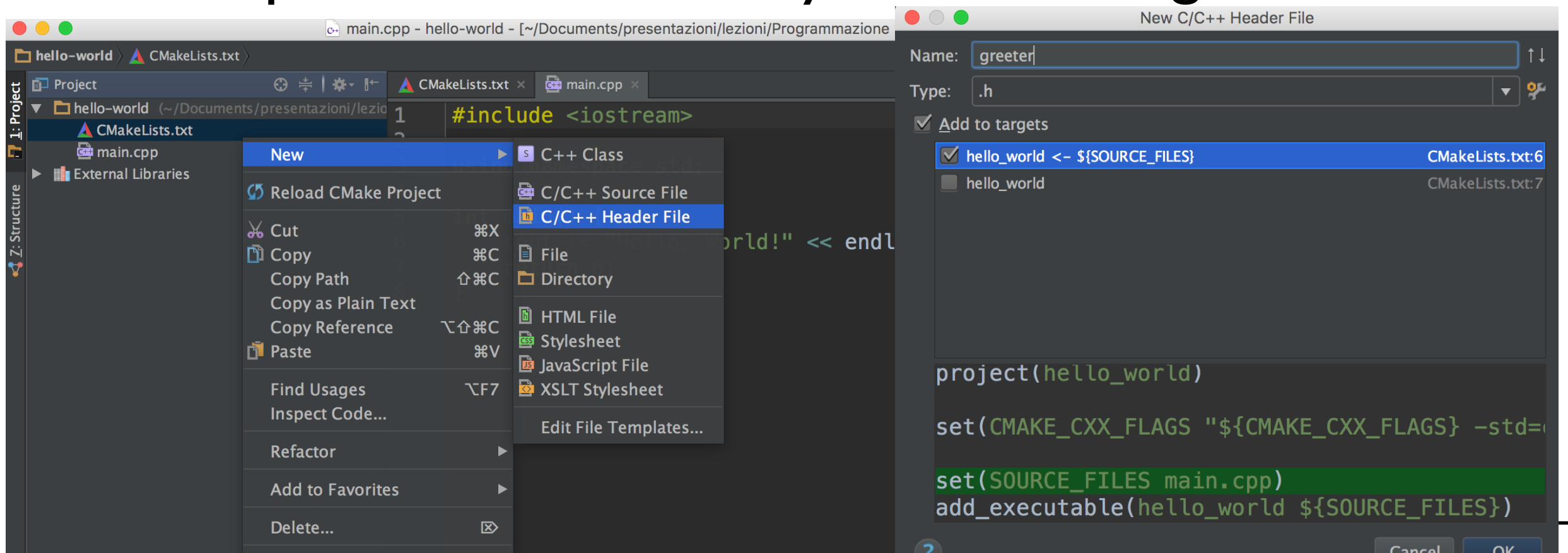
- When you install CLion it will discover if there is the required compiler:
- OS X: LLVM with Clang (most recent)
- Windows: MINGW GCC
- Linux: GCC
- CLion has its own debugger. You can change the selected tools.





# Add a .cpp and .h files

- Add, for example a .h file that contains a function to greet a user, given his name, and add the prototype in the include
- if the include is generated by CLion, it will provide automatically the #define guards





# Add a .cpp and .h files

- Add, for example a .h file that contains a function to greet a user, given his name, and add the prototype in the include
- if the include is generated by CLion, it will provide automatically the #define guards

The screenshot shows the CLion IDE interface. The top window title is "greeter.h - hello-world - [~/Documents/presentazioni/lezioni/Programmazione 2015-2016]". The left sidebar shows the project structure with "greeter.h" selected. The main editor window displays the following code:

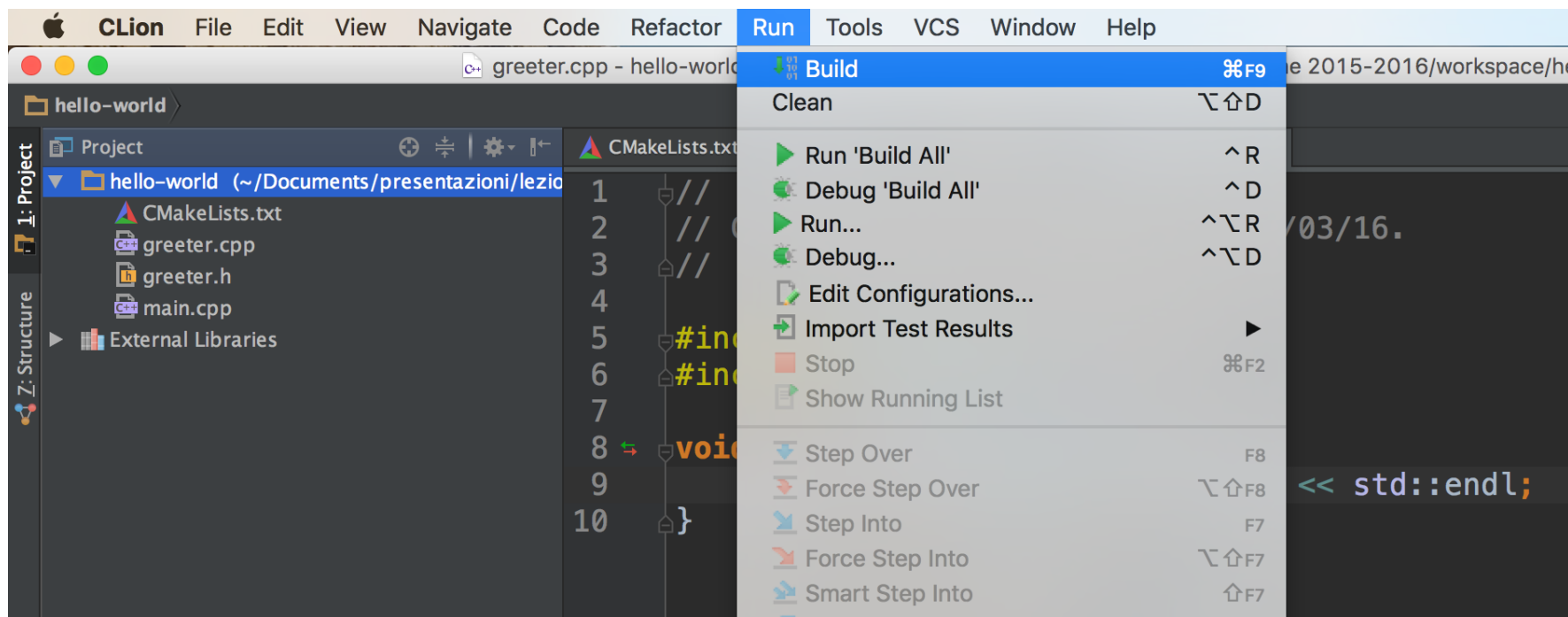
```
1 //  
2 // Created by Marco Bertini on 02/03/16.  
3 //  
4  
5 #ifndef HELLO_WORLD_GREETER_H  
6 #define HELLO_WORLD_GREETER_H  
7  
8 #endif //HELLO_WORLD_GREETER_H  
9
```





# Compile

- Let's say the code has been written in the .cpp (including all the includes required, e.g. `iostream` and the `greeter.h`): compile using Run > Build or using the toolbar icon.
- Check the compile errors (shown in the message panel)







# Compile

- Let's say the code has been written in the .cpp (including all the includes required, e.g. `iostream` and the `greeter.h`): compile using Run > Build or using the toolbar icon.
- Check the compile errors (shown in the message panel)

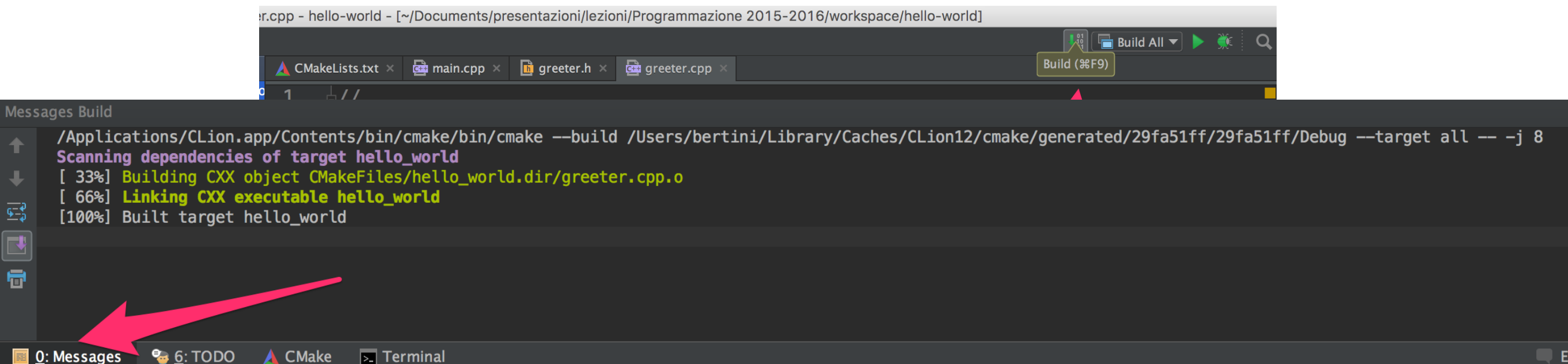
A screenshot of a code editor window. The title bar shows the file path: "r.cpp - hello-world - [~/Documents/presentazioni/lezioni/Programmazione 2015-2016/workspace/hello-world]". The editor has several tabs open: "CMakeLists.txt", "main.cpp", "greeter.h", and "greeter.cpp". The main editor area shows C++ code with line numbers 1 through 10. A toolbar at the top right contains icons for "Build All", a play button, and a search icon. A yellow tooltip labeled "Build (⌘F9)" is visible over the play button icon, with a red arrow pointing to it. The code in the editor is as follows:

```
1 //  
2 // Created by Marco Bertini on 02/03/16.  
3 //  
4  
5 #include "greeter.h"  
6 #include <iostream>  
7  
8 void greet(std::string name) {  
9     std::cout << "Hello " << name << std::endl;  
10 }
```



# Compile

- Let's say the code has been written in the .cpp (including all the includes required, e.g. `iostream` and the `greeter.h`): compile using Run > Build or using the toolbar icon.
- Check the compile errors (shown in the message panel)





# Compile errors

- Don't panic
- Start reading (carefully) the messages from the first to the last. Solve the first errors, perhaps they have an influence on the others.
- In the example the first error is in the `.cpp`

# Compile errors - cont.



greeter.cpp - hello-world - [~/Documents/presentazioni/lezioni/Programmazione 2015-2016/workspace/hello-world]

```
1 //
2 // Created by Marco Bertini on 02/03/16.
3 //
4
5 #include "greeter.h"
6 #include <iostream>
7
8 void greet(string name) {
9     std::cout << "Hello " << name << std::endl;
10 }
```

Messages Build

```
/Applications/CLion.app/Contents/bin/cmake/bin/cmake --build /Users/bertini/Library/Caches/CLion12/cmake/generated/29fa51ff/29fa51ff/Debug --target all -- -j 8
Scanning dependencies of target hello_world
[ 33%] Building CXX object CMakeFiles/hello_world.dir/greeter.cpp.o
/Users/bertini/Documents/presentazioni/lezioni/Programmazione 2015-2016/workspace/hello-world/greeter.cpp:8:12: error: unknown type name 'string'; did you mean 'std::string'?
void greet(string name) {
      ~~~~~
      std::string
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../include/c++/v1/iosfwd:194:65: note: 'std::string' declared here
typedef basic_string<char, char_traits<char>, allocator<char> > string;
      ^
1 error generated.
make[2]: *** [CMakeFiles/hello_world.dir/greeter.cpp.o] Error 1
make[1]: *** [CMakeFiles/hello_world.dir/all] Error 2
make: *** [all] Error 2
```

Build failed in 1s 367ms

0: Messages 6: TODO CMake Terminal Event Log



# Compile errors - cont.

CLion shows in the editor where there's a problem

```
1 //
2 // Created by Marco Bertini on 02/03/16.
3 //
4
5 #include "greeter.h"
6 #include <iostream>
7
8 void greet(string name) {
9     std::cout << "Hello " << name << std::endl;
10 }
```

```
Messages Build
/Applications/CLion.app/Contents/bin/cmake/bin/cmake --build /Users/bertini/Library/Caches/CLion12/cmake/generated/29fa51ff/29fa51ff/Debug --target all -- -j 8
Scanning dependencies of target hello_world
[ 33%] Building CXX object CMakeFiles/hello_world.dir/greeter.cpp.o
/Users/bertini/Documents/presentazioni/lezioni/Programmazione 2015-2016/workspace/hello-world/greeter.cpp:8:12: error: unknown type name 'string'; did you mean 'std::string'?
void greet(string name) {
      ~~~~~
      std::string
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../include/c++/v1/iosfwd:194:65: note: 'std::string' declared here
typedef basic_string<char, char_traits<char>, allocator<char> > string;
1 error generated.
make[2]: *** [CMakeFiles/hello_world.dir/greeter.cpp.o] Error 1
make[1]: *** [CMakeFiles/hello_world.dir/all] Error 2
make: *** [all] Error 2
```

Build failed in 1s 367ms

0: Messages 6: TODO CMake Terminal

Event Log



# Compile errors - cont.

CLion shows in the editor where there's a problem

```
1 //
2 // Created by Marco Bertini on 02/03/16.
3 //
4
5 #include "greeter.h"
6 #include <iostream>
7
8 void greet(string name) {
9     std::cout << "Hello " << name << std::endl;
10 }
```

```
Messages Build
/Applications/CLion.app/Contents/bin/cmake/bin/cmake --build /Users/bertini/Library/Caches/CLion12/cmake/generated/29fa51ff/29fa51ff/Debug --target all -- -j 8
Scanning dependencies of target hello_world
[ 33%] Building CXX object CMakeFiles/hello_world.dir/greeter.cpp.o
/Users/bertini/Documents/presentazioni/lezioni/Programmazione 2015-2016/workspace/hello-world/greeter.cpp:8:12: error: unknown type name 'string'; did you mean 'std::string'?
void greet(string name) {
      ~~~~~
           std::string
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../include/c++/v1/iosfwd:194:65: note: 'std::string' declared here
typedef basic_string<char, char_traits<char>, allocator<char> > string;

1 error generated.
make[2]: *** [CMakeFiles/hello_world.dir/greeter.cpp.o] Error 1
make[1]: *** [CMakeFiles/hello_world.dir/all] Error 2
make: *** [all] Error 2
```

Read the message: the type string is unknown. The compiler even suggests the correct type. Error messages will vary depending on the compiler though !



# Compile errors - cont.

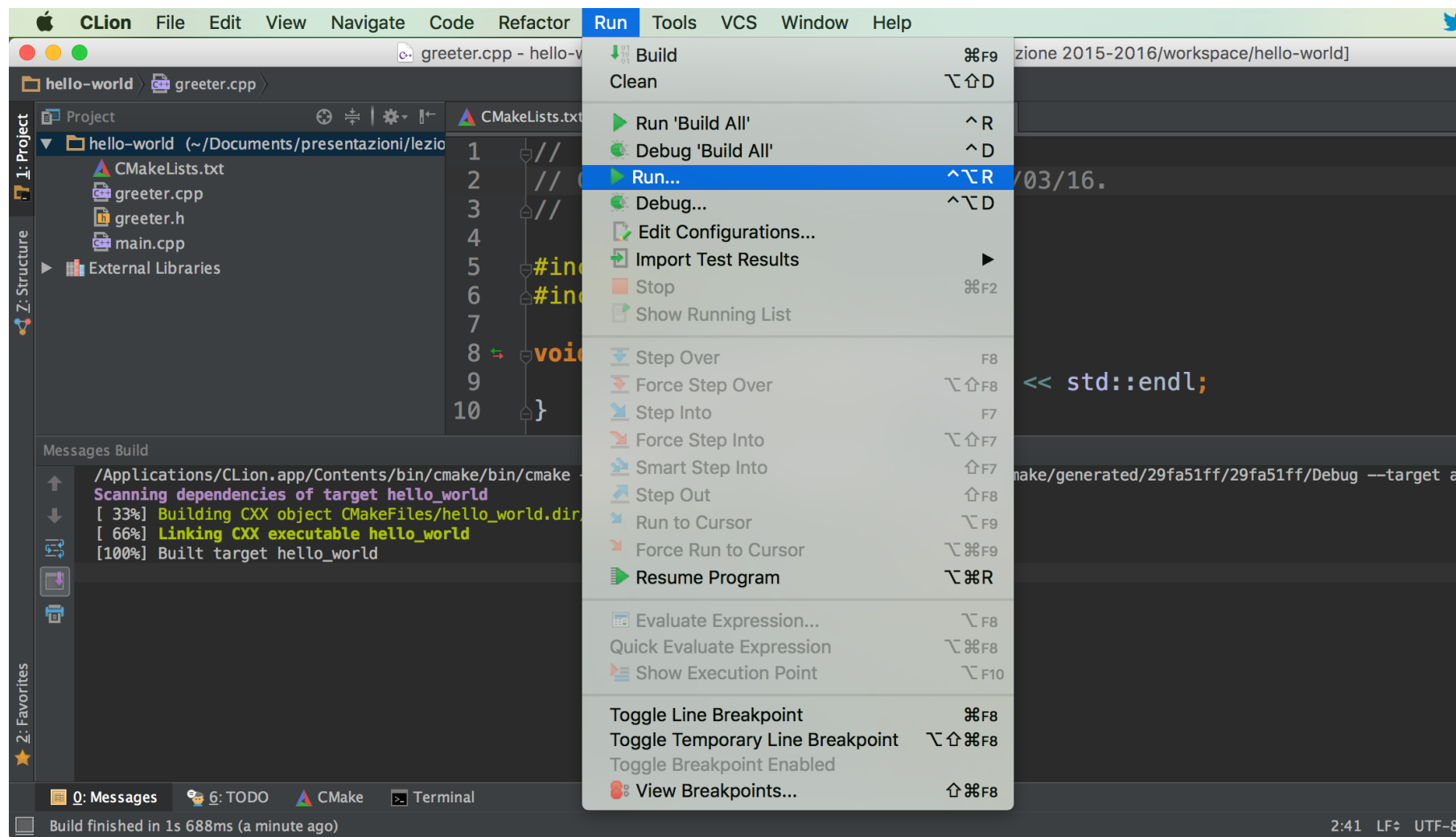
- Correct the error: in this case it was necessary to add `std::` to `string` (we are not using “`using namespace std;`” in this file !)
- Build again to check the correction





# Run the program

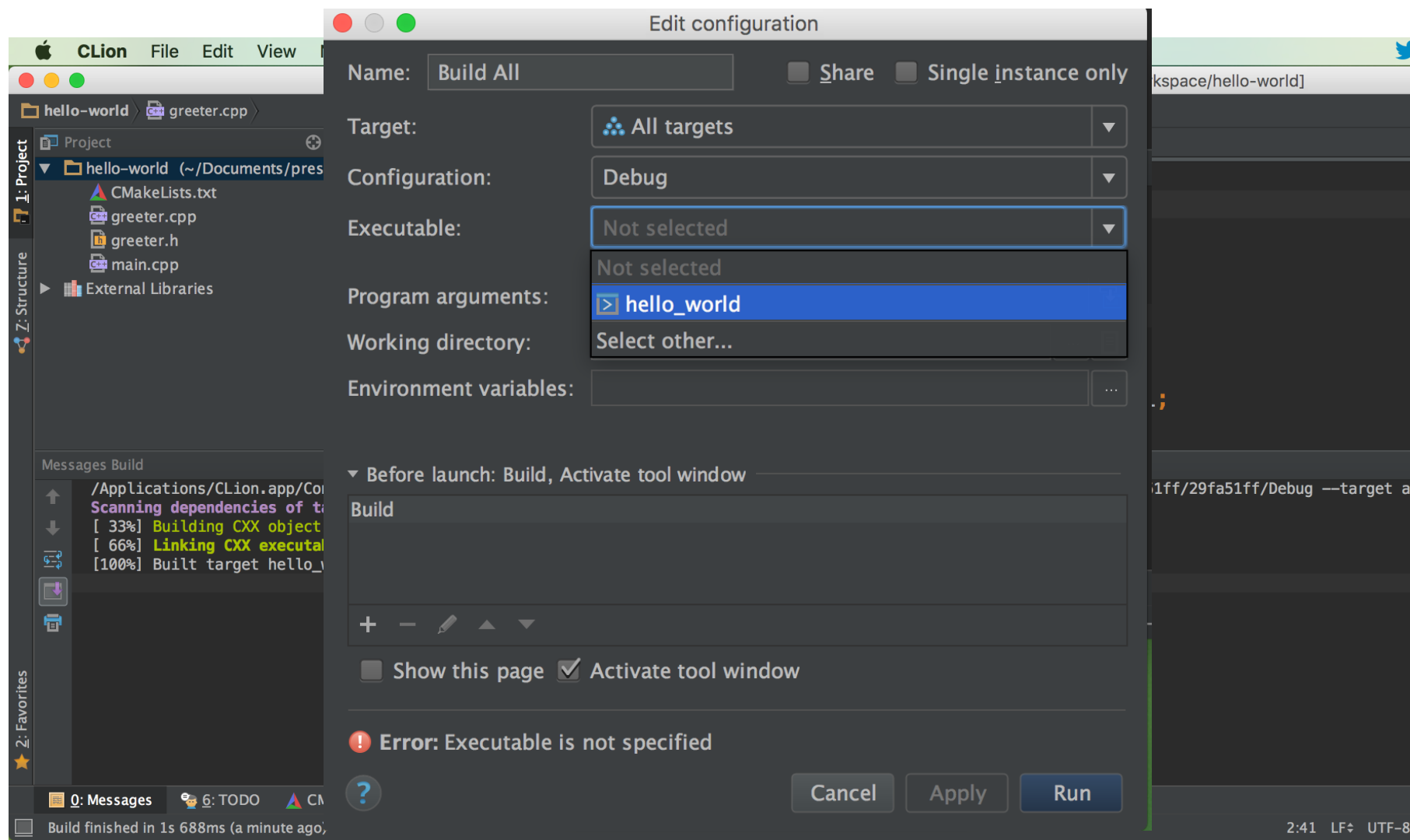
- Use the menu Run > Run... > Select the executable to run (a project may have more than one).  
Later on the program will appear in the Run History





# Run the program

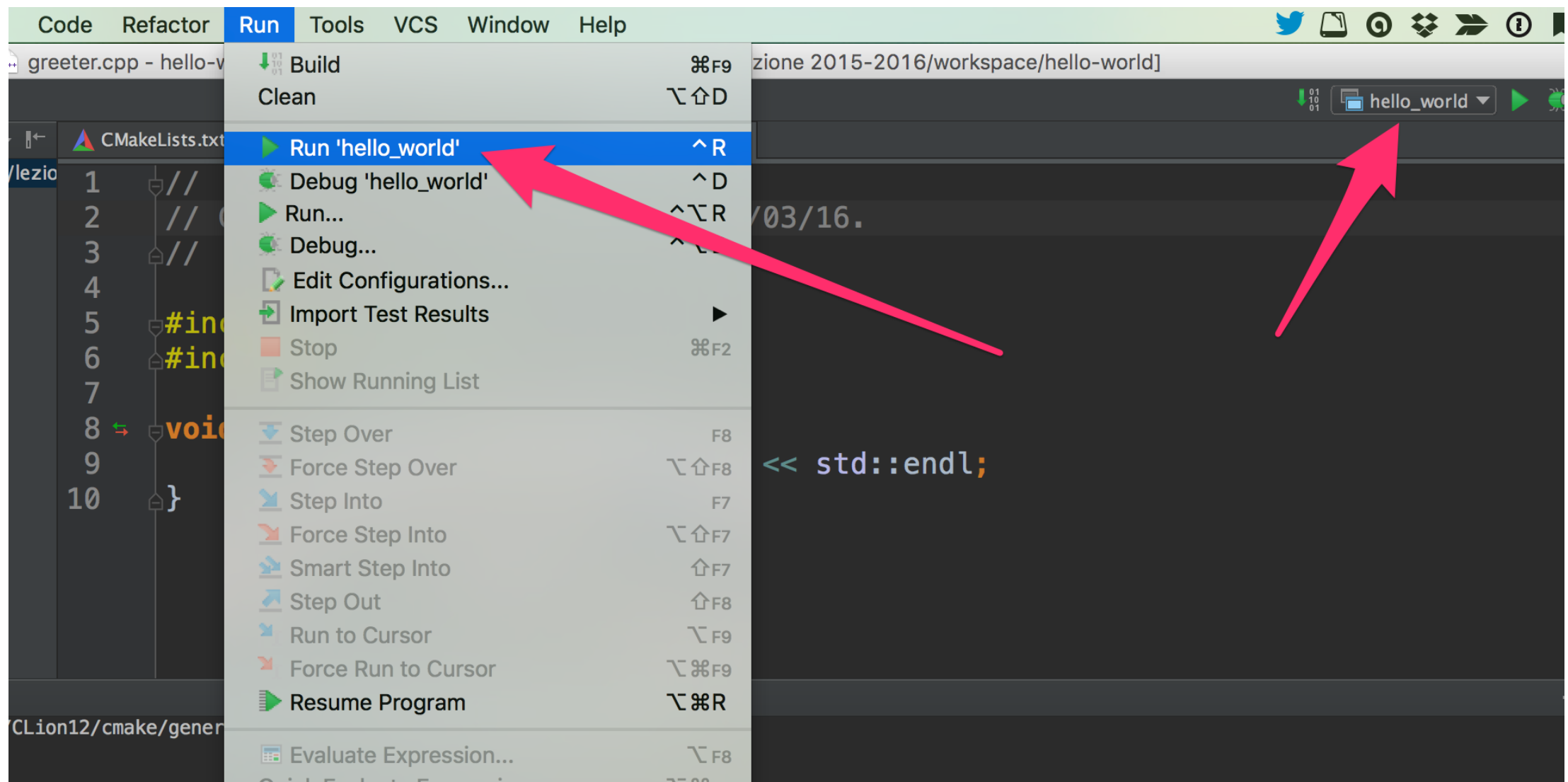
- Use the menu Run > Run... > Select the executable to run (a project may have more than one).  
Later on the program will appear in the Run History





# Run the program

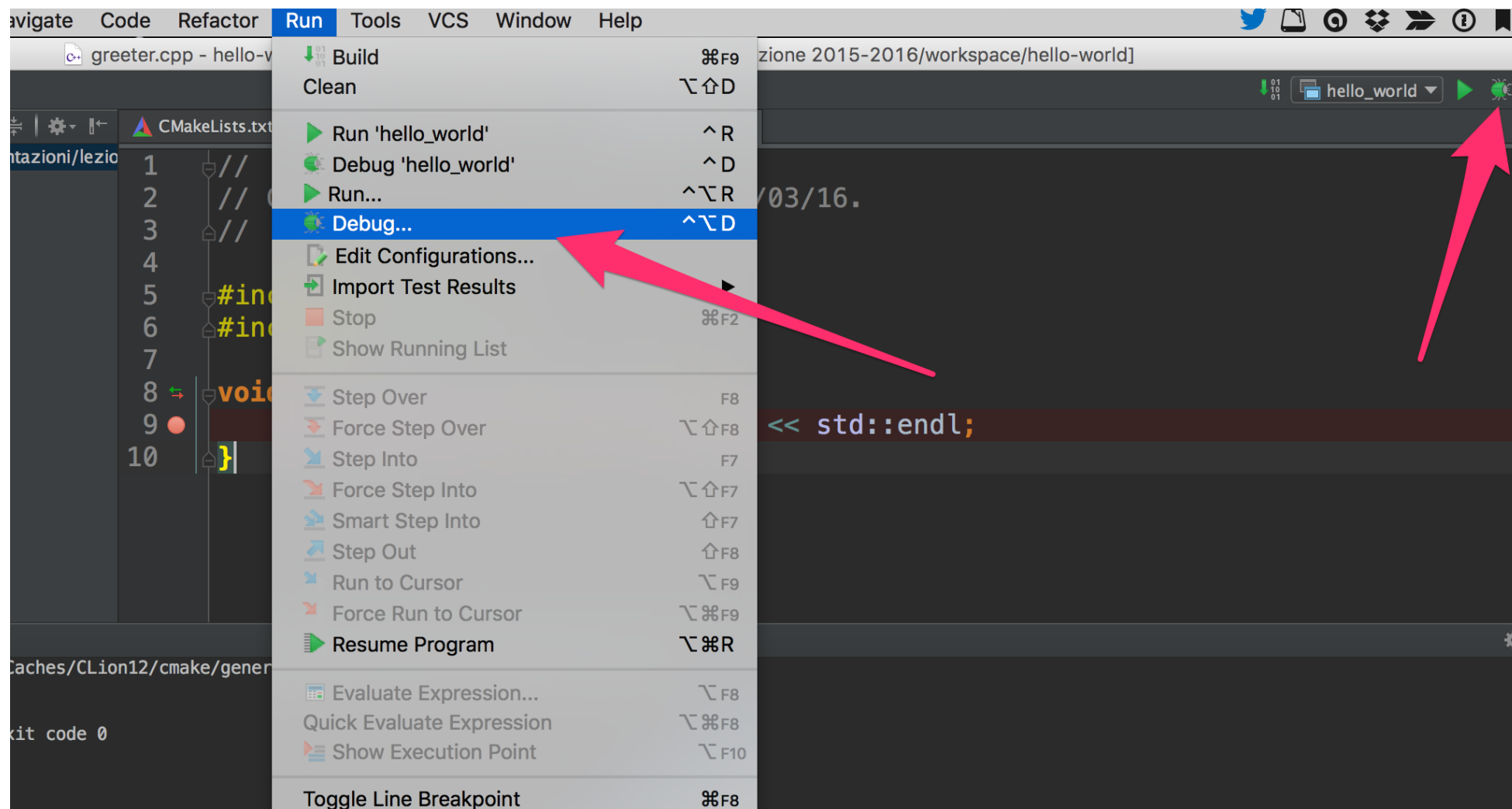
- Use the menu Run > Run... > Select the executable to run (a project may have more than one).  
Later on the program will appear in the Run History





# Debug the program

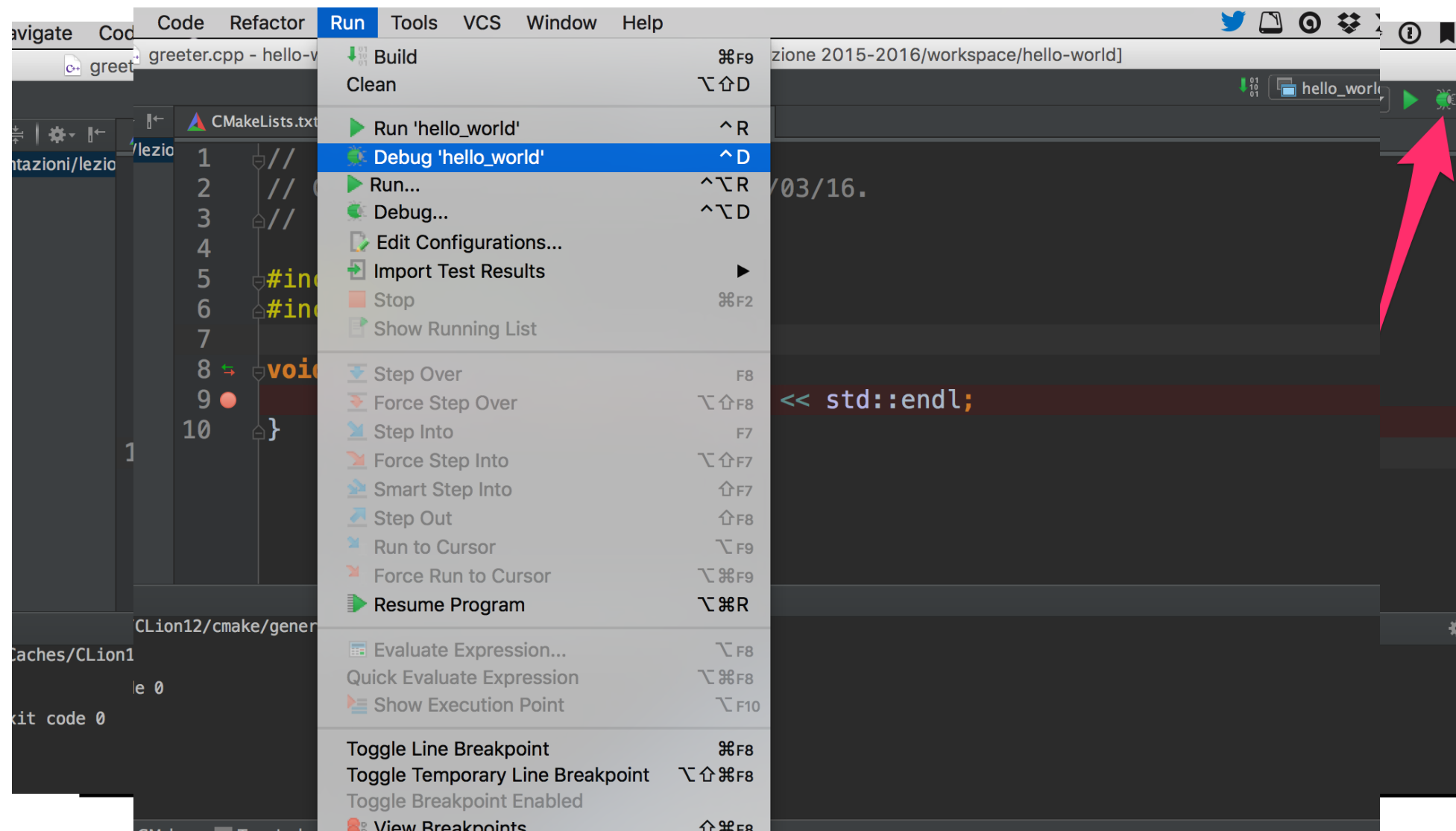
- In order to debug the program must be compiled so that additional information, useful for the debugger, is added to the files
- Add a breakpoint in CLion (click the left side of the line), then execute the program in the debugger (Run > Debug application)





# Debug the program

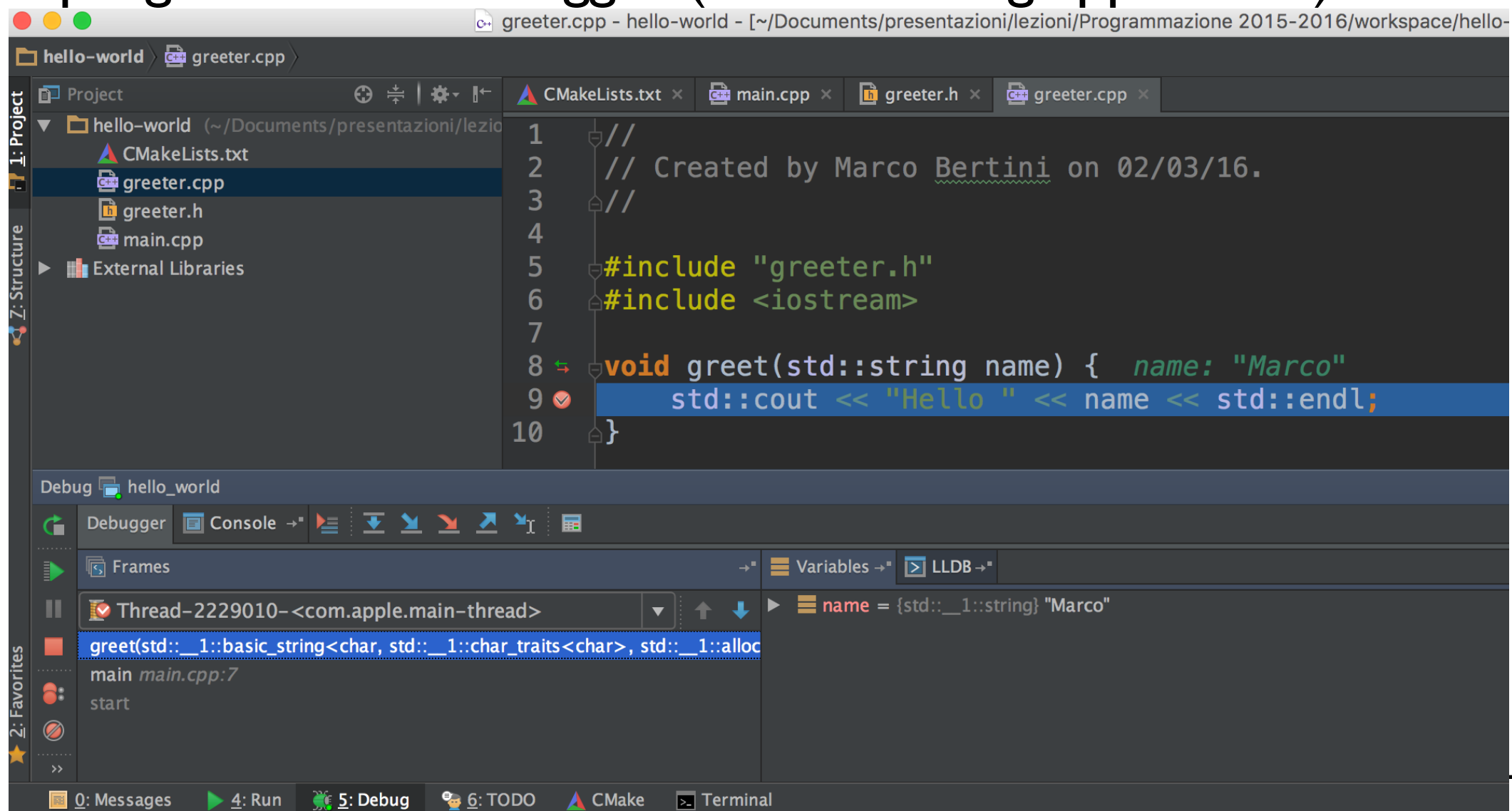
- In order to debug the program must be compiled so that additional information, useful for the debugger, is added to the files
- Add a breakpoint in CLion (click the left side of the line), then execute the program in the debugger (Run > Debug application)





# Debug the program

- In order to debug the program must be compiled so that additional information, useful for the debugger, is added to the files
- Add a breakpoint in CLion (click the left side of the line), then execute the program in the debugger (Run > Debug application)







# Some style guidelines

- There are a plethora of C++ coding style recommendations, sometimes even contradictory.
  - Two very good recommendations:
    1. Any violation to the guidelines is allowed if it enhances readability.
    2. The rules can be violated if there are strong personal objections against them.
-





# Naming conventions

- Names representing types must be in mixed case starting with upper case: follow this rule when writing classes.
  - Variable names must be in mixed case starting with lower case (like Java).
  - Names representing methods or functions must be verbs and written in mixed case starting with lower case (like Java).
-



# Naming conventions - cont.

- Names representing namespaces should be all lowercase.
- All names should be written in English.



# Files

- C++ header files should have the extension `.h` (preferred) or `.hpp`. Source files can have the extension `.c++`, `.C`, `.cc` or `.cpp`.
  - A class should be declared in a header file and defined in a source file where the name of the files match the name of the class.
  - Header files must contain an include guard.
  - Include statements must be located at the top of a file only.
-