



FreedomBytes

2023

CMake: An introduction to build, test and package software

- La domanda fondamentale: Che cos'è CMake?
- Facciamo ordine: da dove cominciare?
- Compilazione programmi C/C++: make files e standard convetions

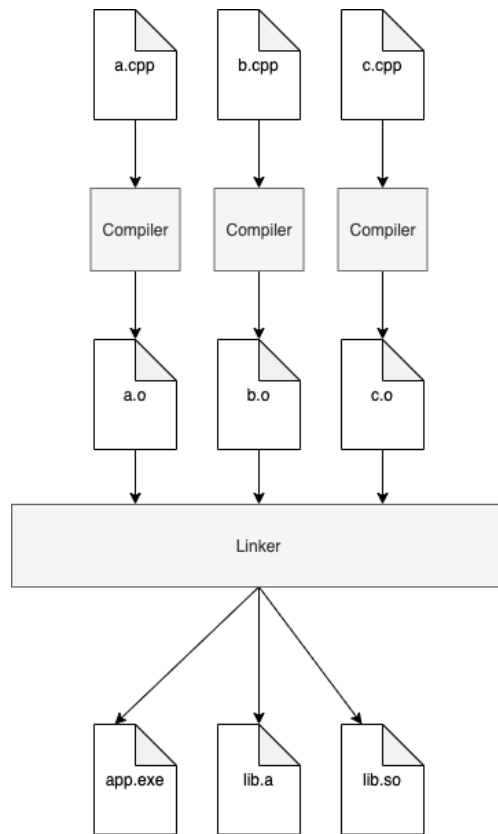
Che cos'è CMake?

- CMake is the **de-facto standard** for building C++ code.
- CMake is an extensible, open-source system that manages the **build process** in an operating system and in a compiler-independent manner.
- CMake is a **collection of open-source and cross-platform tools** used to **build and distribute** software.
- CMake is known as a **meta build system**. It doesn't actually build your source code: instead, it generates native project files for the target platform.



Iniziamo: Build

- **Progetto:** insieme di file di testo
- **Compilatore:** trasforma ogni file in codice oggetto
- **Linker:** collega tutto insieme
- **Risultato:**
 - Eseguibile
 - Libreria statica
 - Libreria dinamica



Build in practice

```
> g++ -c a.cpp -o a.o
```

```
> g++ -c b.cpp -o b.o
```

```
> g++ -c c.cpp -o c.o
```

```
> g++ -c main.cpp -o main.o
```

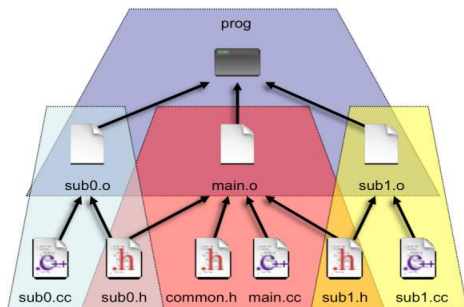
```
> g++ a.o b.o c.o main.o -o main
```

```
> ./main
```



CLI Commands

MakeFile



```
all: sub0.o sub1.o main.o
    mpicxx sub0.o sub1.o main.o -o prog

sub0.o: sub0.cc sub0.h
    mpicxx -c sub0.cc

sub1.o: sub1.cc sub1.h
    mpicxx -c sub1.cc

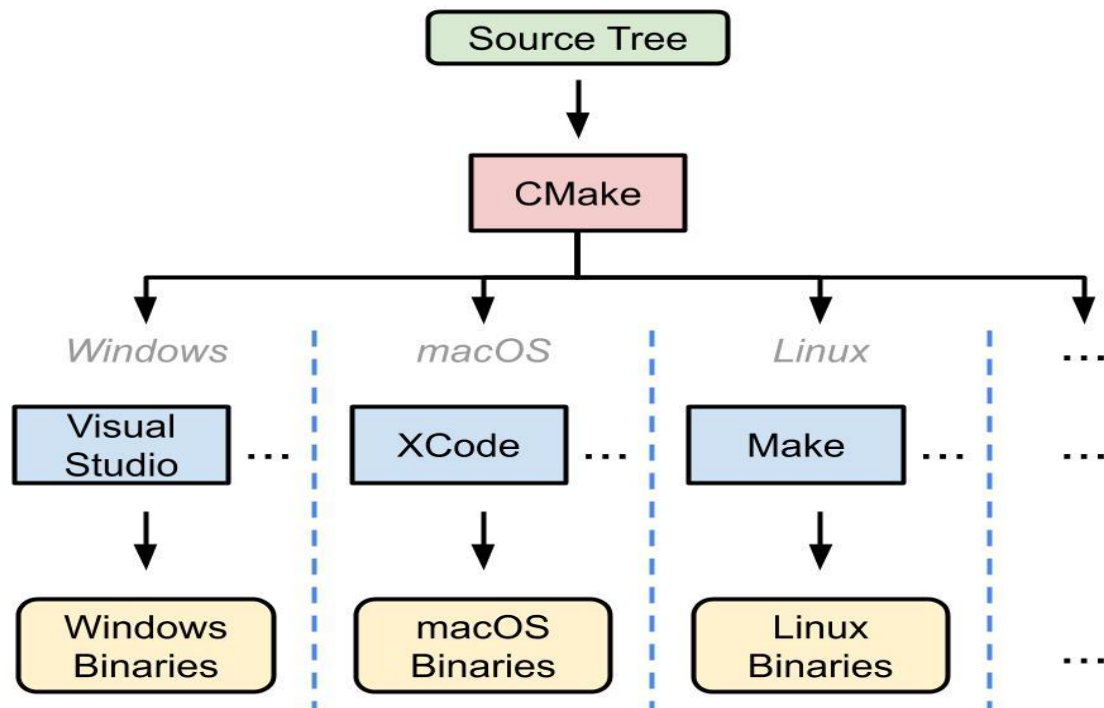
main.o: main.cc common.h sub0.h sub1.h
    mpicxx -c main.cc

clean:
    rm *.o prog
```

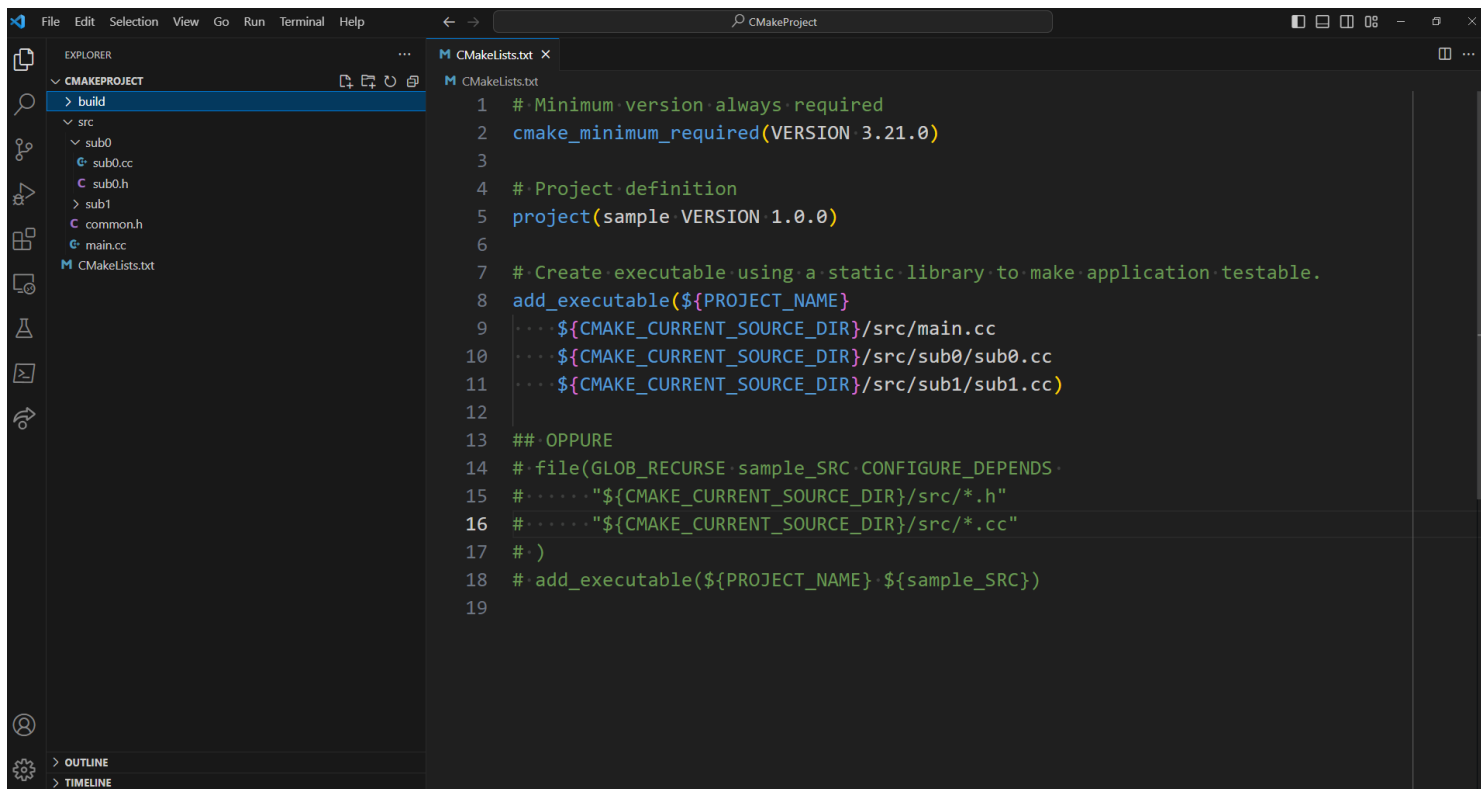
- Difficile da mantenere
- Dipendenza dal compilatore: se non con boilerplate complessi
- Dipendenza dal Sistema: unix
- Linkare librerie esterne può diventare complicato



Arriviamo a CMake



Primo Esempio



The screenshot shows the Visual Studio Code interface with a CMake project. The Explorer panel on the left shows the project structure: a 'build' directory, a 'src' directory containing 'sub0' and 'sub1' subdirectories, and a 'common.h' file. The main editor displays the 'CMakeLists.txt' file with the following content:

```
1 # Minimum version always required
2 cmake_minimum_required(VERSION 3.21.0)
3
4 # Project definition
5 project(sample VERSION 1.0.0)
6
7 # Create executable using a static library to make application testable.
8 add_executable(${PROJECT_NAME}
9     ...${CMAKE_CURRENT_SOURCE_DIR}/src/main.cc
10    ...${CMAKE_CURRENT_SOURCE_DIR}/src/sub0/sub0.cc
11    ...${CMAKE_CURRENT_SOURCE_DIR}/src/sub1/sub1.cc)
12
13 ## OPPURE
14 # file(GLOB_RECURSE sample_SRC CONFIGURE_DEPENDS
15 #     ... "${CMAKE_CURRENT_SOURCE_DIR}/src/*.h"
16 #     ... "${CMAKE_CURRENT_SOURCE_DIR}/src/*.cc"
17 # )
18 # add_executable(${PROJECT_NAME} ${sample_SRC})
19
```



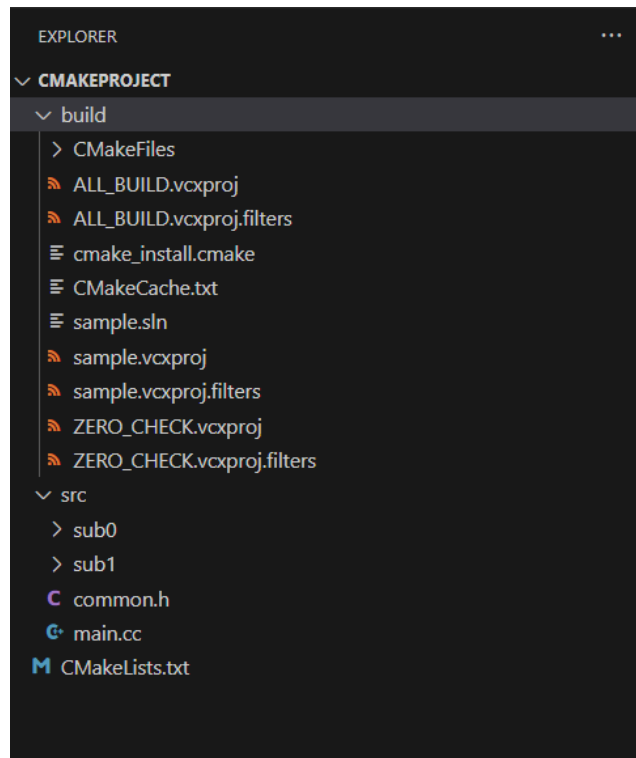
Comandi

Generare i file nella cartella attuale

> cmake .

Generare i file nella cartella 'build':

> cmake -B build



ATTENZIONE



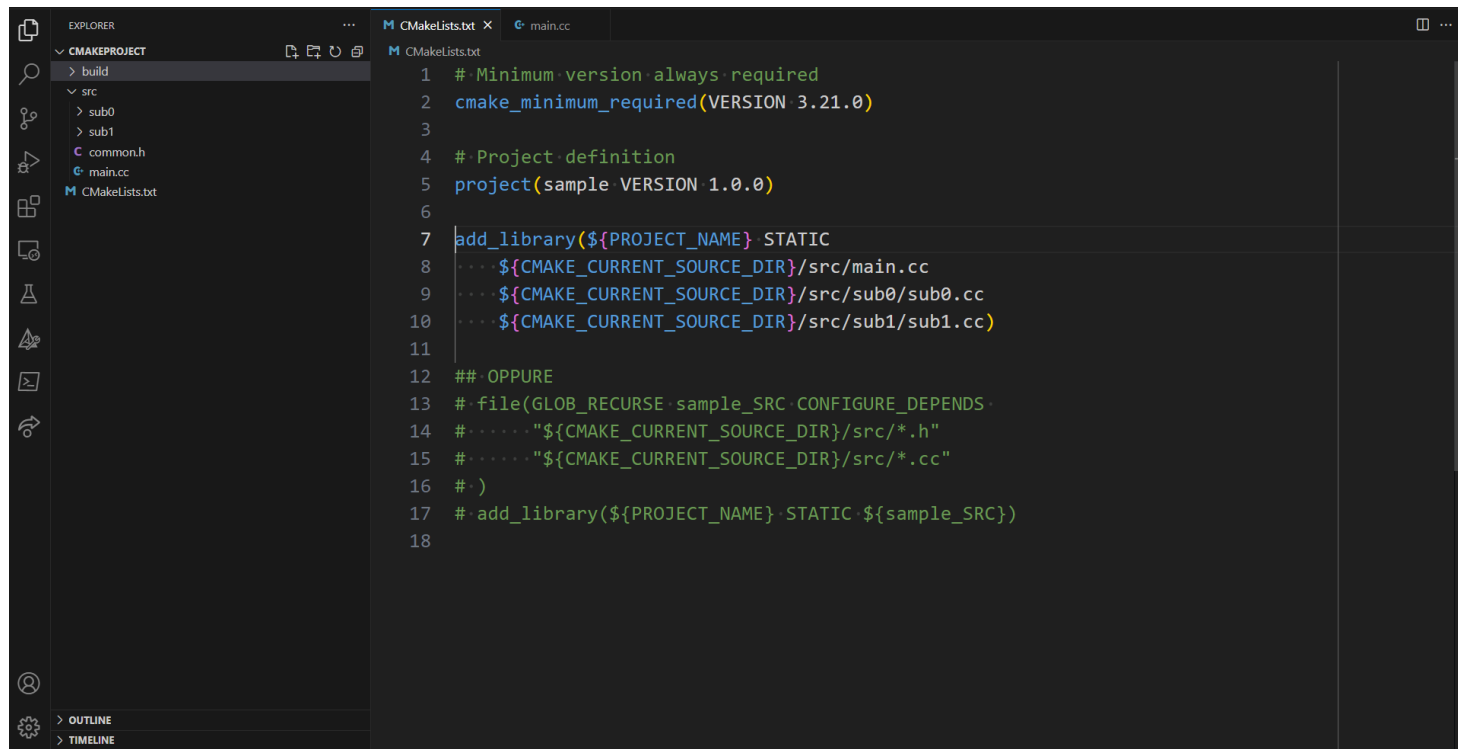
- Cosa succede se modifico un file?
- Cosa succede se aggiungo un file?
- Cosa succede se elimino un file?

Target

- Eseguibile
- Libreria Statica
- Libreria Dinamica
- Libreria Header Only
- `add_executable(<name>)`
- `add_library(<name> STATIC)`
- `add_library(<name> SHARED)`
- `add_library(<name> INTERFACE)`



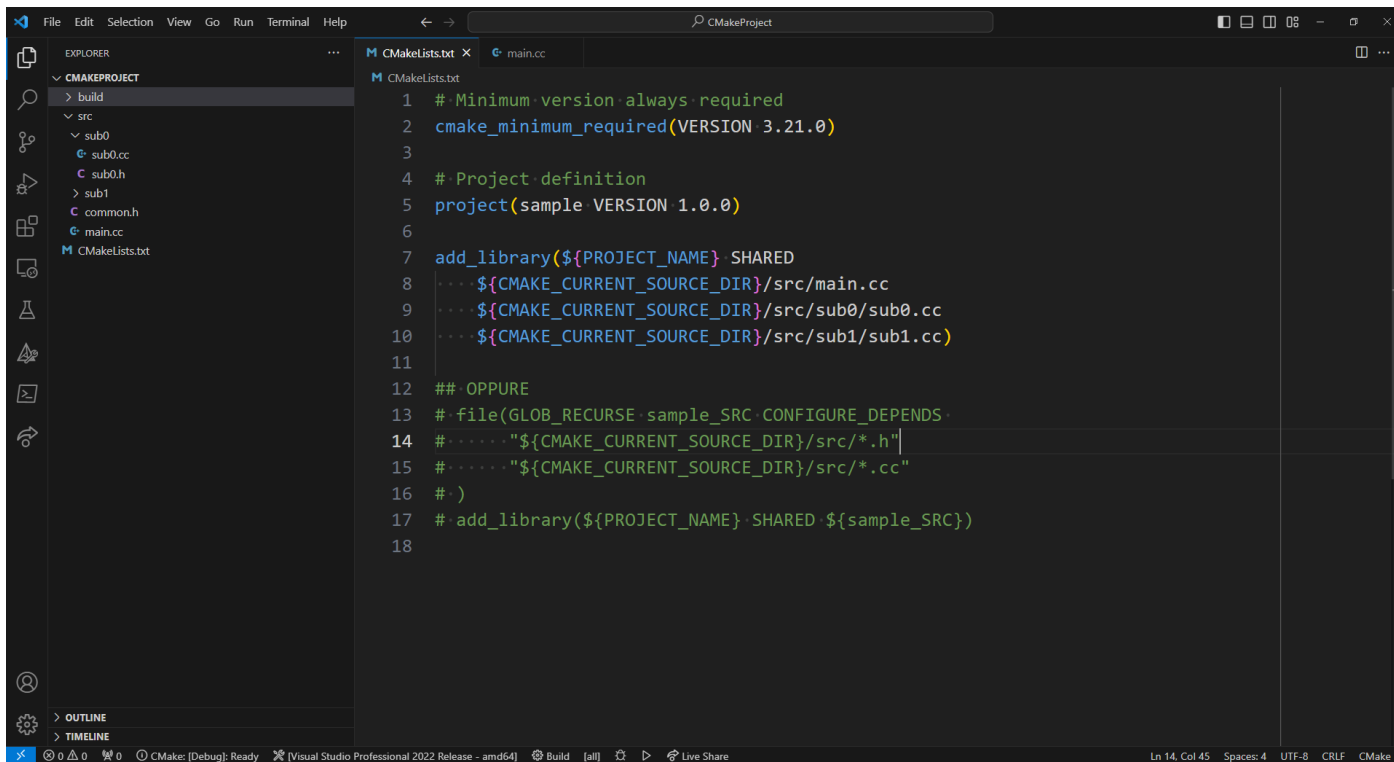
Libreria Statica



```
1 # Minimum version always required
2 cmake_minimum_required(VERSION 3.21.0)
3
4 # Project definition
5 project(sample VERSION 1.0.0)
6
7 add_library(${PROJECT_NAME} STATIC
8     ${CMAKE_CURRENT_SOURCE_DIR}/src/main.cc
9     ${CMAKE_CURRENT_SOURCE_DIR}/src/sub0/sub0.cc
10    ${CMAKE_CURRENT_SOURCE_DIR}/src/sub1/sub1.cc)
11
12 ## OPURE
13 # file(GLOB_RECURSE sample_SRC CONFIGURE_DEPENDS
14 #     "${CMAKE_CURRENT_SOURCE_DIR}/src/*.h"
15 #     "${CMAKE_CURRENT_SOURCE_DIR}/src/*.cc"
16 # )
17 # add_library(${PROJECT_NAME} STATIC ${sample_SRC})
18
```



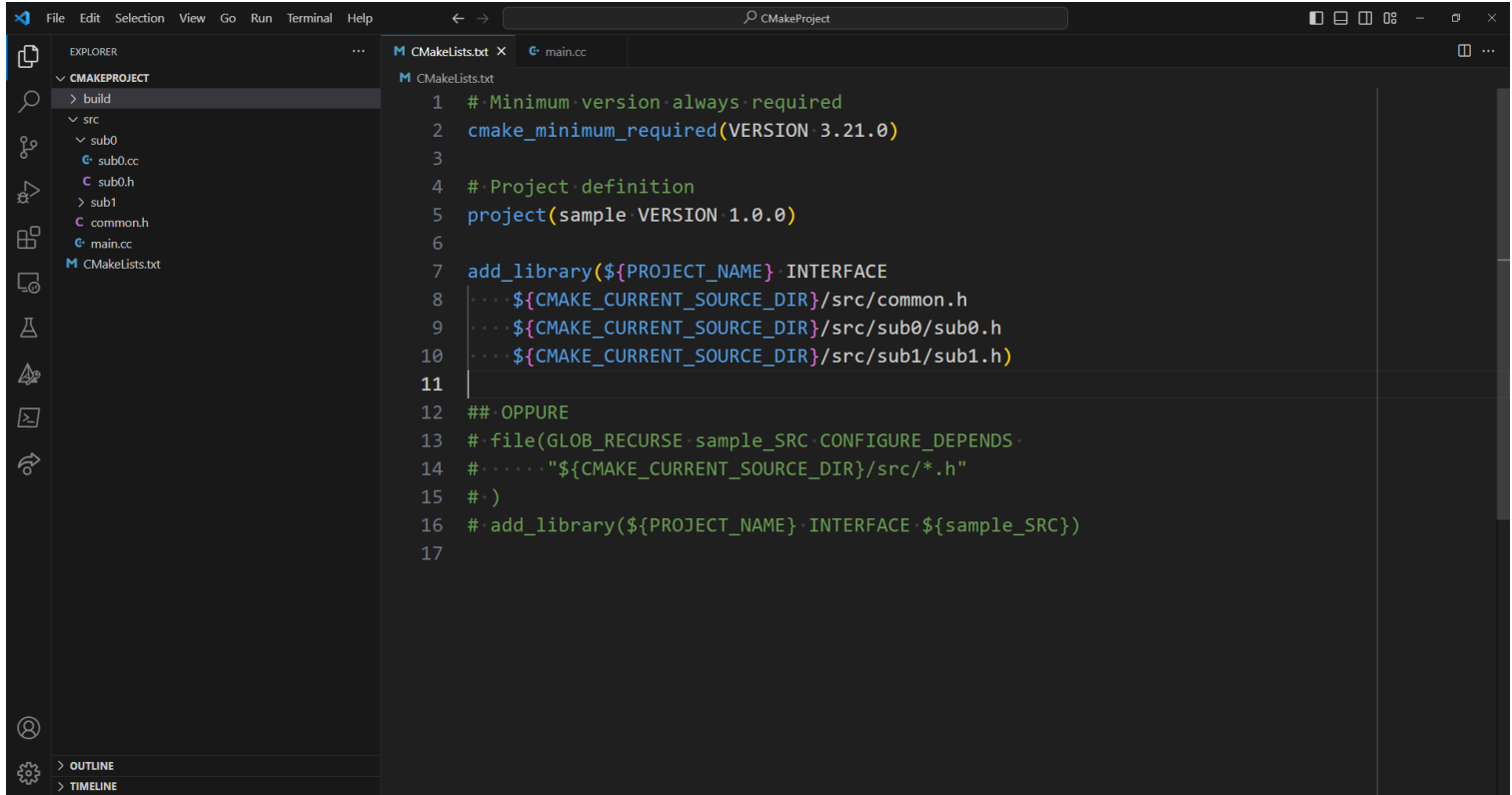
Libreria Dinamica



```
1 # Minimum version always required
2 cmake_minimum_required(VERSION 3.21.0)
3
4 # Project definition
5 project(sample VERSION 1.0.0)
6
7 add_library(${PROJECT_NAME} SHARED
8     ... ${CMAKE_CURRENT_SOURCE_DIR}/src/main.cc
9     ... ${CMAKE_CURRENT_SOURCE_DIR}/src/sub0/sub0.cc
10    ... ${CMAKE_CURRENT_SOURCE_DIR}/src/sub1/sub1.cc)
11
12 ## OPPURE
13 # file(GLOB_RECURSE sample_SRC CONFIGURE_DEPENDS
14 #     ... "${CMAKE_CURRENT_SOURCE_DIR}/src/*.h"
15 #     ... "${CMAKE_CURRENT_SOURCE_DIR}/src/*.cc"
16 # )
17 # add_library(${PROJECT_NAME} SHARED ${sample_SRC})
18
```



Libreria Header Only



The screenshot shows a Visual Studio Code editor window with a CMake project. The Explorer sidebar on the left shows the project structure: a 'build' directory, a 'src' directory containing 'sub0' (with 'sub0.cc' and 'sub0.h') and 'sub1' (with 'common.h'), and a 'main.cc' file. The main editor area displays the 'CMakeLists.txt' file with the following content:

```
1 # Minimum version always required
2 cmake_minimum_required(VERSION 3.21.0)
3
4 # Project definition
5 project(sample VERSION 1.0.0)
6
7 add_library(${PROJECT_NAME} INTERFACE
8     ...${CMAKE_CURRENT_SOURCE_DIR}/src/common.h
9     ...${CMAKE_CURRENT_SOURCE_DIR}/src/sub0/sub0.h
10    ...${CMAKE_CURRENT_SOURCE_DIR}/src/sub1/sub1.h)
11
12 ## OPUPRE
13 # file(GLOB_RECURSE sample_SRC CONFIGURE_DEPENDS
14 #     ... "${CMAKE_CURRENT_SOURCE_DIR}/src/*.h"
15 # )
16 # add_library(${PROJECT_NAME} INTERFACE ${sample_SRC})
17
```



Usare una libreria: cosa serve

- La libreria da linkare: bisogna dirlo al linker
- La cartella dove andare a cercare la libreria da linkare
- La cartella dove andare a cercare gli header



Include header: dobbiamo dirlo noi

- **Modo 1:**

```
# The include directories are added to the INCLUDE_DIRECTORIES directory
# property for the current CMakeLists file. They are also added to the
# INCLUDE_DIRECTORIES target property for each target in the current CMakeLists
# file. The target property values are the ones used by the generators.
include_directories(PUBLIC "include")
# The DIRECTORY form installs contents of one or more directories to a given
# destination. The directory structure is copied verbatim to the destination.
install(DIRECTORY "include" DESTINATION .)
```

- **Modo 2: Generator properties**

```
target_include_directories(${PROJECT_NAME} PUBLIC
    "$<BUILD_INTERFACE:${CMAKE_CURRENT_LIST_DIR}/include>"
    "$<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>")
```



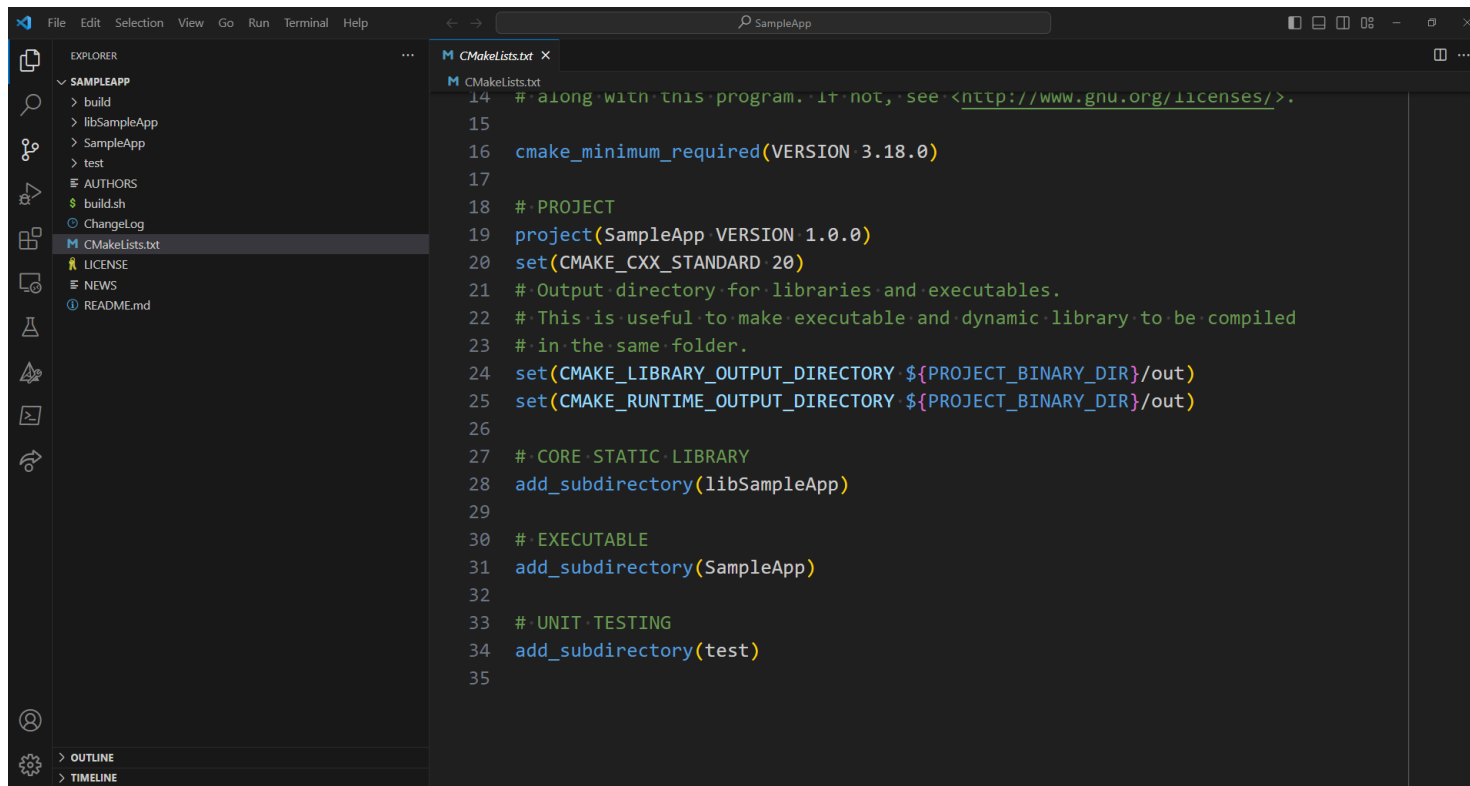
Linkare una libreria

```
target_link_libraries(  
    <target_name>  
    PRIVATE|PUBLIC|INTERFACE <library_name1>  
    PRIVATE|PUBLIC|INTERFACE <library_name1>  
    ...  
  
    PRIVATE|PUBLIC|INTERFACE <library_nameN>)
```

- **PRIVATE:** la dipendenza e' visibile solo nei file sorgenti
- **PUBLIC:** la dipendenza e' visibile sia nei file sorgenti che negli header
- **INTERFACE:** la dipendenza e' visibile solo nei file header



Struttura Applicazione



```
14 # along with this program. If not, see <http://www.gnu.org/licenses/>.
15
16 cmake_minimum_required(VERSION 3.18.0)
17
18 # PROJECT
19 project(SampleApp VERSION 1.0.0)
20 set(CMAKE_CXX_STANDARD 20)
21 # Output directory for libraries and executables.
22 # This is useful to make executable and dynamic library to be compiled
23 # in the same folder.
24 set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/out)
25 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/out)
26
27 # CORE STATIC LIBRARY
28 add_subdirectory(libSampleApp)
29
30 # EXECUTABLE
31 add_subdirectory(SampleApp)
32
33 # UNIT TESTING
34 add_subdirectory(test)
35
```



libSampleApp

```
# Create application core as static library
add_library(lib${PROJECT_NAME} STATIC "")

file(GLOB lib${PROJECT_NAME}_SRC CONFIGURE_DEPENDS "src/*.cc")

target_sources(lib${PROJECT_NAME} PRIVATE ${lib${PROJECT_NAME}_SRC})

# Public interface
target_include_directories(lib${PROJECT_NAME} PUBLIC
    "$<BUILD_INTERFACE:${CMAKE_CURRENT_LIST_DIR}/include>"
    "$<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>")
```



SampleApp

```
add_executable(${PROJECT_NAME})  
  
file(GLOB ${PROJECT_NAME}_SRC CONFIGURE_DEPENDS "*.cc")  
  
target_sources(${PROJECT_NAME} PRIVATE ${${PROJECT_NAME}_SRC})  
  
target_link_libraries(${PROJECT_NAME} PRIVATE lib${PROJECT_NAME})
```



SampleAppTest

```
add_executable(${PROJECT_NAME}Test)

file(GLOB ${PROJECT_NAME}_TEST_SRC CONFIGURE_DEPENDS "src/*.cc")

target_sources(${PROJECT_NAME}Test PRIVATE ${${PROJECT_NAME}_TEST_SRC})

target_link_libraries(${PROJECT_NAME}Test PRIVATE lib${PROJECT_NAME})
```





- If/else
- Loop
- Include cmake files
- Package system
- Etc..



Ricapitolando

- Multiplatforma
- Semplifica operazioni laboriose: linking
- Gestione del Progetto as folder
- Si integra con diversi IDE ed editor
- Ben supportato da conan





Domande?