

Processo lato utente



Lato utente

- Come utenti possiamo trovarci di fronte a un progetto gestito con CMake
- Scaricare un sorgente gestito con CMake permette di avere, oltre al sorgente:
 - La descrizione del progetto
 - Un sistema che predisponde la compilazione il software di CMake
 - Un sistema per gestire la compilazione selettiva dei moduli
 - Parti del progetto possono essere compilate o meno a seconda delle impostazioni



Lato utente

- Esempio: scarichiamo un progetto che ha la struttura:

Rational

```
└── include
    └── rational.h
└── CMakeLists.txt
└── src
    ├── main.cpp
    └── rational.cpp
```

Due attività fondamentali

Configurazione



Compilazione



Lato utente

- Esempio: scarichiamo un progetto che ha la struttura:

```
Rational
├── include
│   └── rational.h
├── CMakeLists.txt
└── src
    ├── main.cpp
    └── rational.cpp
└── build
```

Configurazione:

- Creazione directory build
- Configurazione tramite comando cmake

Compilazione:

- Comando make

È il comando che permette di fare la configurazione. Ogni volta che si modifica CMakeLists.txt bisogna ridare il comando cmake ..

```
user@host:~/Rational/$ mkdir build
user@host:~/Rational/$ cd build
user@host:~/Rational/build/$ cmake ..
user@host:~/Rational/build/$ make
```

Tool Linux che permette di gestire un progetto fatto con makefile



Lato utente

- Usare la directory build come output è molto utile:
 - Non mescola output di compilazione con il progetto sorgente
 - Permette di ritornare alla situazione iniziale eliminando la directory

Processo lato sviluppatore



Lato sviluppatore

- Si crea una descrizione del progetto nello standard CMake
 - Scrivere a mano i CMakeLists.txt
 - Usare un IDE per generarli automaticamente
 - Più complicati
 - In genere, non devono essere modificati
 - Possono essere sovrascritti in automatico



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Lato sviluppatore

- Creiamo un CMakeLists.txt



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
```



Versione minima di
CMake (2.84 è
solitamente OK)

In genere si usa la versione più
vecchia che supporta le
funzionalità necessarie



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
```



Compilatore
configurato per
C++11
Equivalent al flag
-std=c++11



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
project(rational)
```

Nome del progetto
Accessibile come
\${PROJECT_NAME}





Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
project(rational)

find_package(Qt5 REQUIRED COMPONENTS Core)
```



Librerie esterne:
find_package() le
cerca nel sistema e
le linka

Per i progetti che faremo noi, che
useranno solo la libreria standard,
questa riga non è necessaria



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
project(rational)

find_package(Qt5 REQUIRED COMPONENTS Core)

include_directories(include)
```



Directory in cui
cercare file header
(rispetto alla
posizione di
CMakeLists.txt)



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
project(rational)

find_package(Qt5 REQUIRED COMPONENTS Core)

include_directories(include)

add_library(${PROJECT_NAME} SHARED
            src/rational.cpp)
```



Crea la libreria
librational.so
Libreria dinamica
(shared)
A partire dal
sorgente
rational.cpp

È comune descrivere il path
relativo alla root del progetto



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
project(rational)

find_package(Qt5 REQUIRED COMPONENTS Core)

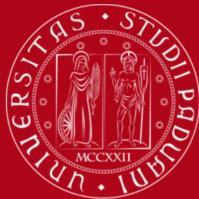
include_directories(include)

add_library(${PROJECT_NAME} SHARED
            src/rational.cpp)

add_executable(main src/main.cpp)
```



Crea l'eseguibile
main a partire dal
sorgente main.cpp



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
project(rational)

find_package(Qt5 REQUIRED COMPONENTS Core)

include_directories(include)

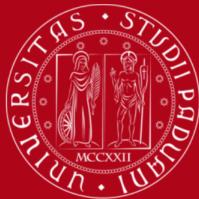
add_library(${PROJECT_NAME} SHARED
            src/rational.cpp)

add_executable(main src/main.cpp)

target_link_libraries(main ${PROJECT_NAME})
```



Linka l'eseguibile
main alla libreria
librational



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
project(rational)

find_package(Qt5 REQUIRED COMPONENTS Core)

include_directories(include)

add_library(${PROJECT_NAME} SHARED
            src/rational.cpp)

add_executable(main src/main.cpp)

target_link_libraries(main ${PROJECT_NAME})

install(TARGETS ${PROJECT_NAME}
        LIBRARY DESTINATION lib
        INCLUDES DESTINATION include
)
```



Installa nel sistema
la libreria e gli
header

Disponibili poi da usare



Lato sviluppatore

- Le librerie template hanno bisogno di un meccanismo diverso
 - Non generano librerie precompilate



Lato sviluppatore

```
cmake_minimum_required(VERSION 2.84)
set(CMAKE_CXX_STANDARD 11)
project(myvector)

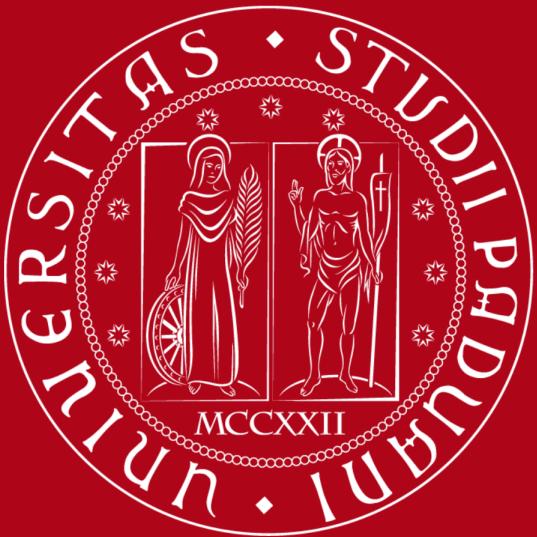
add_library(${PROJECT_NAME} INTERFACE)

target_include_directories(${PROJECT_NAME} INTERFACE
    ${BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include}
    ${BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/src}
)

add_executable(main src/main.cpp)
target_link_libraries(main ${PROJECT_NAME})
```

Libreria interface:
composta di soli
header

Directory in cui
trovare i file
.h/.hpp



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Introduzione a CMake

Stefano Ghidoni