VU C-C++: Build Environment

How to manage projects

Markus Walzthöni

December 10, 2023

UIBK

Overview

- Compilation - Linking
- Libraries
- Make - CMake
- The Template
- Testing

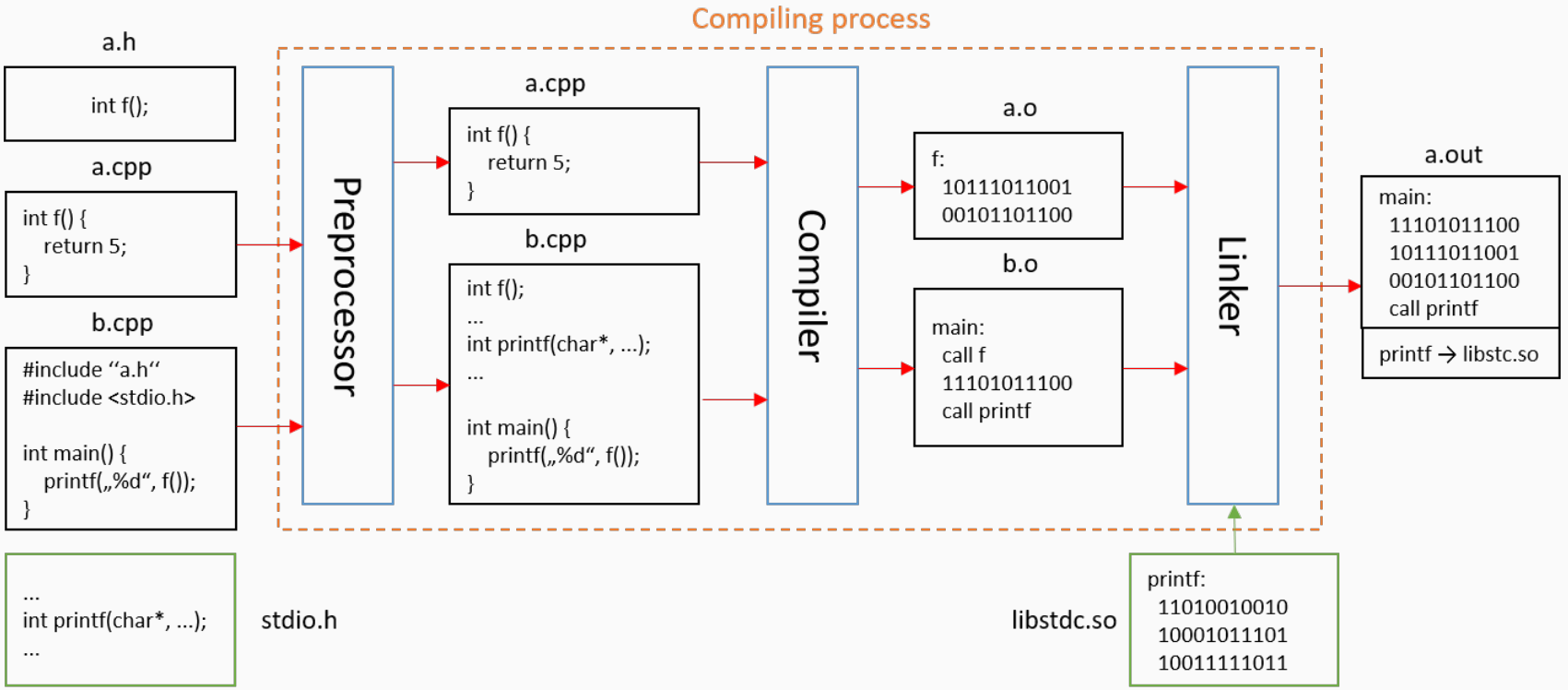
Compilation - Linking

Compilation Recap

Command:

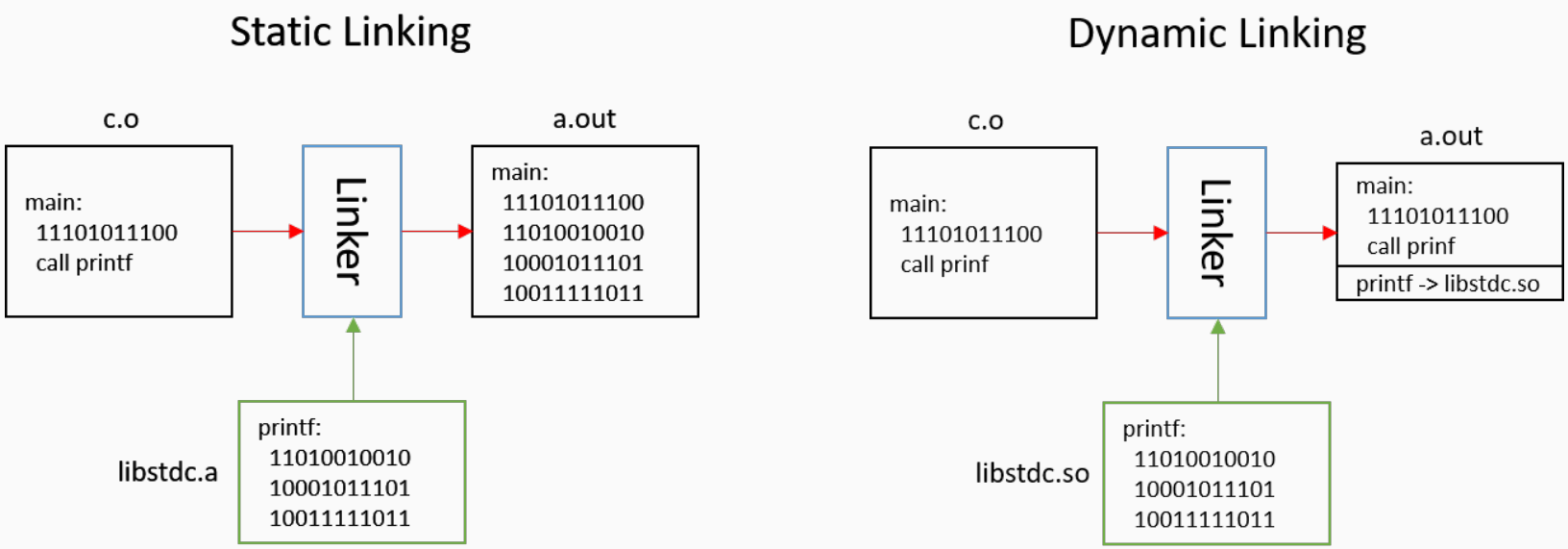
```
gcc a.cpp b.cpp
```

What is happening?



Linking Recap

Types of Linking



Useful Commands

`cpp a.cpp` → get the `a.cpp` after the preprocessor

`gcc -c a.cpp b.cpp` → compiling `a.cpp` and `b.cpp`, get object file `a.o` and `b.o`

`gcc a.cpp -o a.o` → compiling `a.cpp`, get object file `a.o`

`gcc a.cpp -o a` → compiling `a.cpp`, get executable `a`

`gcc a.cpp` → compiling `a.cpp`, get `a.out` as executable

`ldd a.out` → show the libraries a executable `a.out` is linking to

`objdump -d a.o` → prints the object file `a.o` in readable form

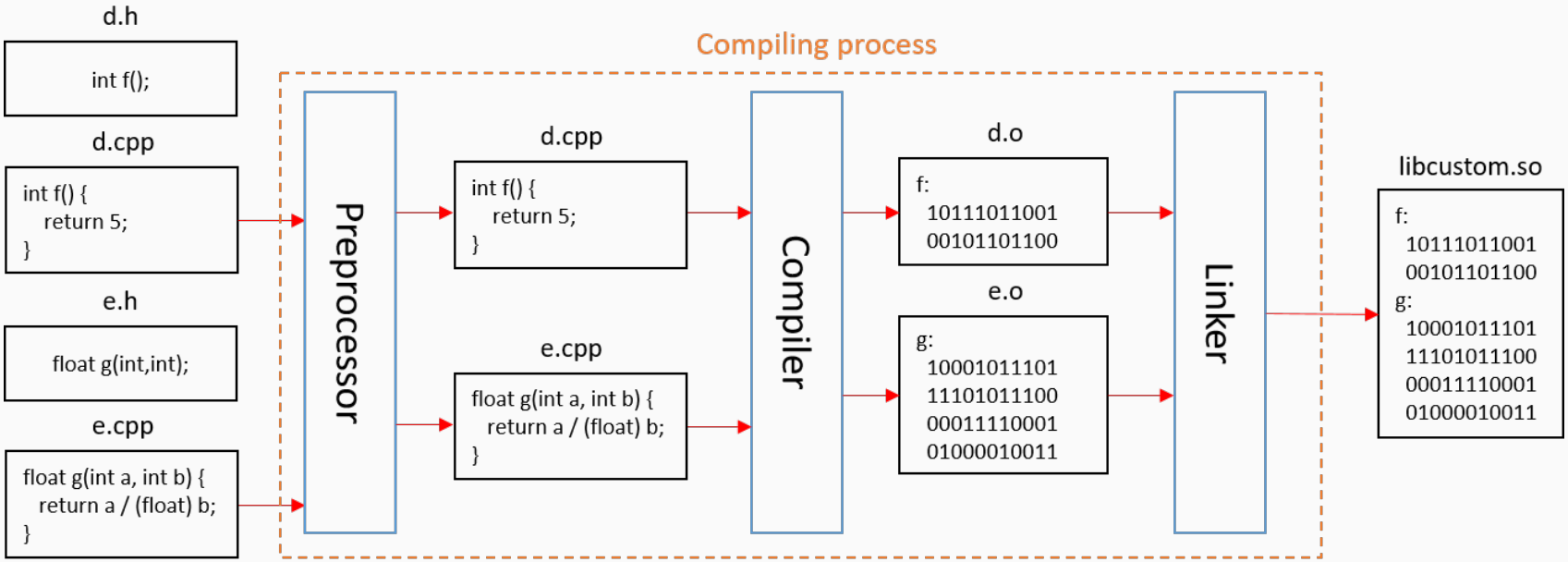
`objdump -d a.out` → prints the executable file `a.o` in readable form

Libraries

How to create a shared library

Command:

```
gcc -shared d.cpp e.cpp -o libcustom.so
```



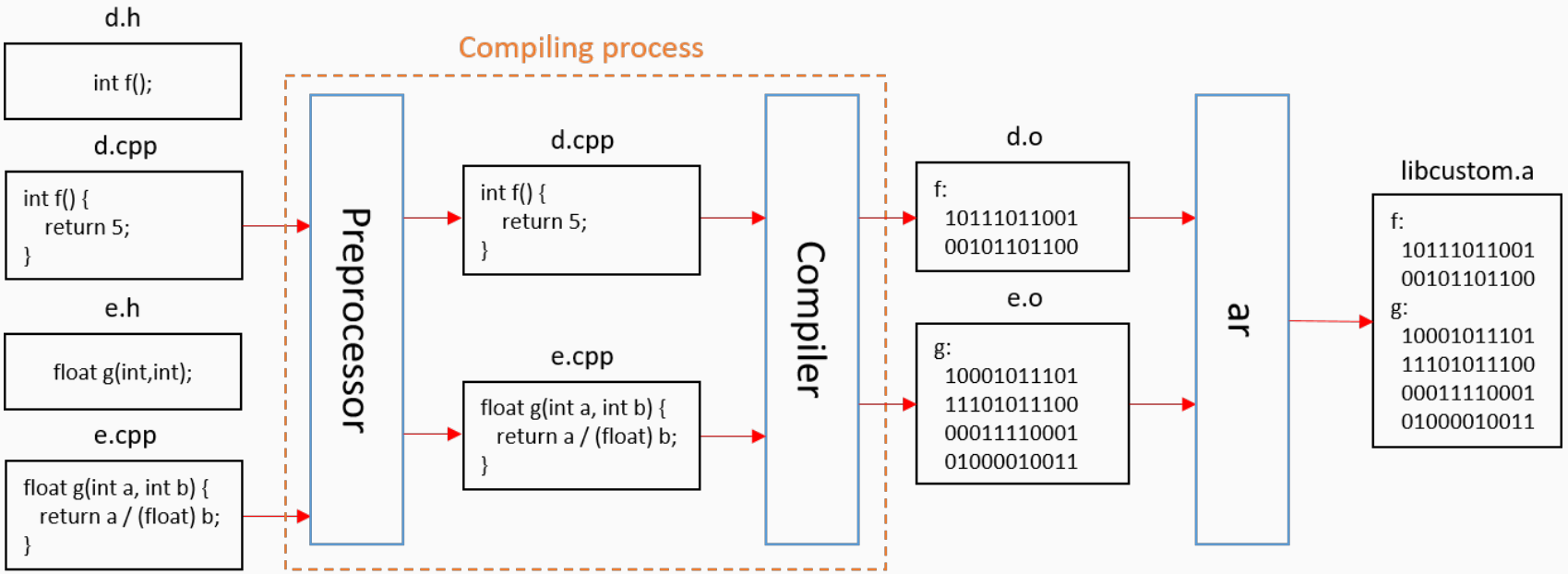
To install the library global copy it e.g. in `/usr/lib/`

Note: a shared library name has to start with `lib` and end with `.so`

How to create a static library

Command:

```
gcc -c a.cpp b.cpp
ar rcs libcustom.a a.o b.o
```



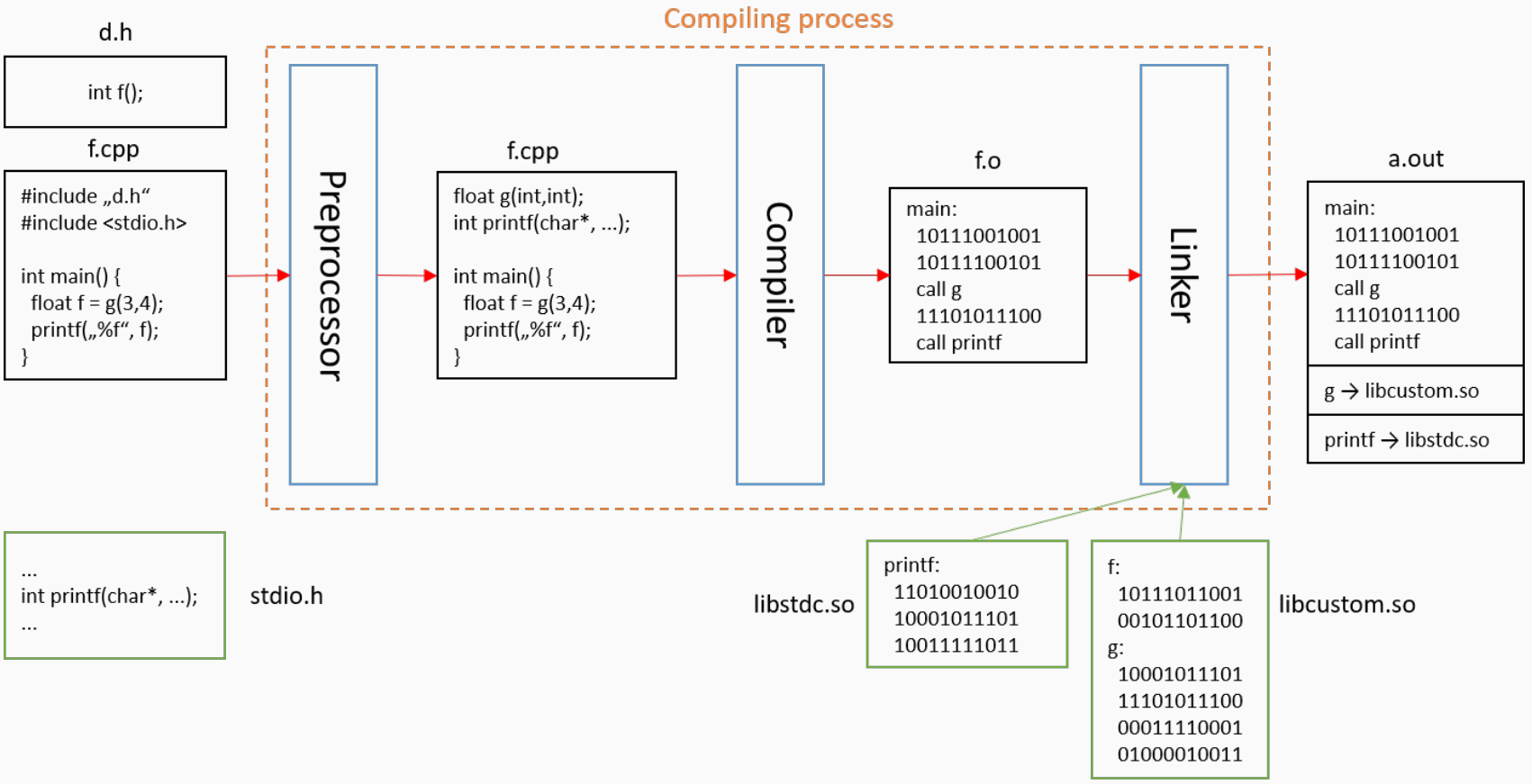
To install the library global copy it e.g. in `/usr/lib/`

Note: a static library name has to start with `lib` and end with `.a`

How to use a custom library

Command:

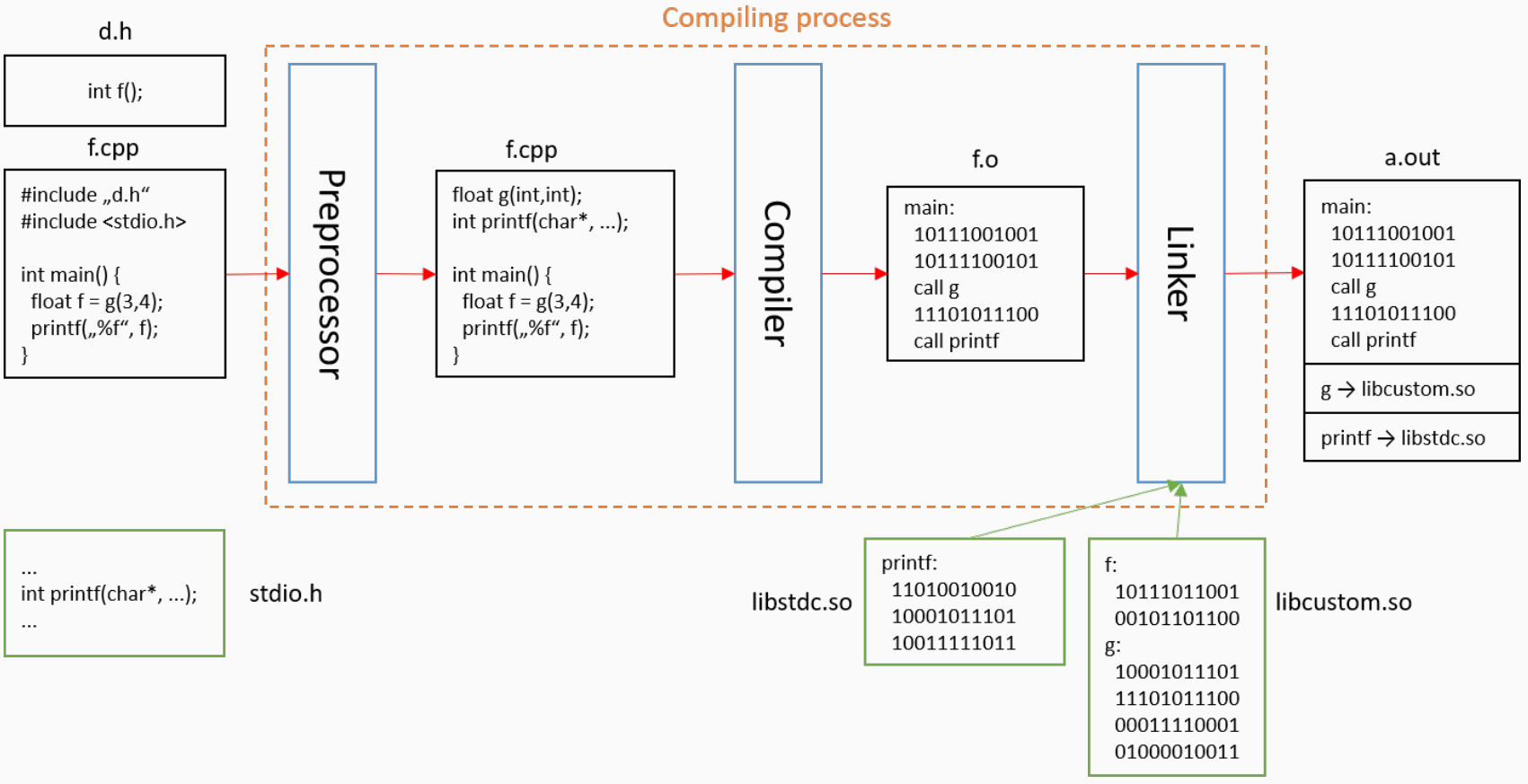
```
gcc f.cpp -lcustom
```



How to use a custom library

If the library is only stored locally you can run this:

```
export LD_LIBRARY_PATH=$pwd
gcc f.cpp libcustom.so
```



Make - CMake

Make

- The basis is a text file called `Makefile`
- This file is managing the compilation and linking calls
- The user can specify compiler flags and to which libraries the compiler has to link a object file or an executable
- Has features, that you can compile single of multiple files at once
- Has feature to clean the whole builded files

CMake

- The basis is a text file called `CMakeLists.txt`
- Basically doing nothing else, than managing and creating your `Makefiles`
- The user can specify compiler flags and to which libraries the compiler hat to link
- currently version 3.28.0 is the latest one available
- downloadable here

CMake - What can you do?

- create executables
- create libraries
- set flags, set instru
- deal with variables / conditions / loops
- check file system
- include packages (e.g. Eigen, gTest, Boost, . . .)

CMake - What can you do?

- create for different build systems (Make, ninja, VS Studio 16, VS Code)
- create various builds (release/debug - different compiler)
- minimal rebuilds (cache)
- cross-platform

CMake - How can you do it?

set up basic project informations

```
# General properties
cmake_minimum_required( VERSION 3.5 )
project( cmake-template )
set( CMAKE_CXX_STANDARD 17 )
set( main_lib ${PROJECT_NAME} )
```

CMake - How can you do it?

loops / variables / conditions

```
  if( UNIX )
    list( APPEND GLOBAL_LINKING_FLAGS "-pthread" )
  endif()

  set( GCC_COMPILE_FLAGS "${GLOBAL_COMPILE_FLAGS}" )
  list( APPEND GCC_COMPILE_FLAGS "-fmax-errors=5" )

  foreach( CHILD ${CHILDREN} )
    if( IS_DIRECTORY ${input_path}/${CHILD}
      AND ${CHILD} MATCHES "^[A-Z]" )
    list( APPEND DIR_LIST ${CHILD} )
  endif()
endforeach( CHILD )
```

CMake - How can you do it?

create execudable

```
add_executable( EXEC ${SOURCE_FILE} )
set_target_properties( EXEC PROPERTIES
  RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/bin" )
target_link_libraries( EXEC ${main_lib} )
install (TARGETS EXEC DESTINATION
  ${CMAKE_INSTALL_PREFIX}/bin )
```

CMake - How can you do it?

cerate library

```
add_library( LIB [SHARED/STATIC] ${SOURCE_FILE} )
set_target_properties( LIB PROPERTIES
  LIBRARY_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/lib" )
target_link_libraries( LIB ${main_lib} )
install (TARGETS LIB DESTINATION
  ${CMAKE_INSTALL_PREFIX}/bin )
```

CMake - How can you do it?

file system

```
file( GLOB FILES_THIS_FOLDER RELATIVE ${input_path}
  ${input_path}/* )
file( GLOB_RECURSE ALL_FILES RELATIVE ${input_path}
  ${input_path}/* )
if( IS_DIRECTORY ${input_path}/${child} )
  list( APPEND dirlist ${child} )
endif()
get_filename_component( ${output_var} ${input_var}
  DIRECTORY )
```

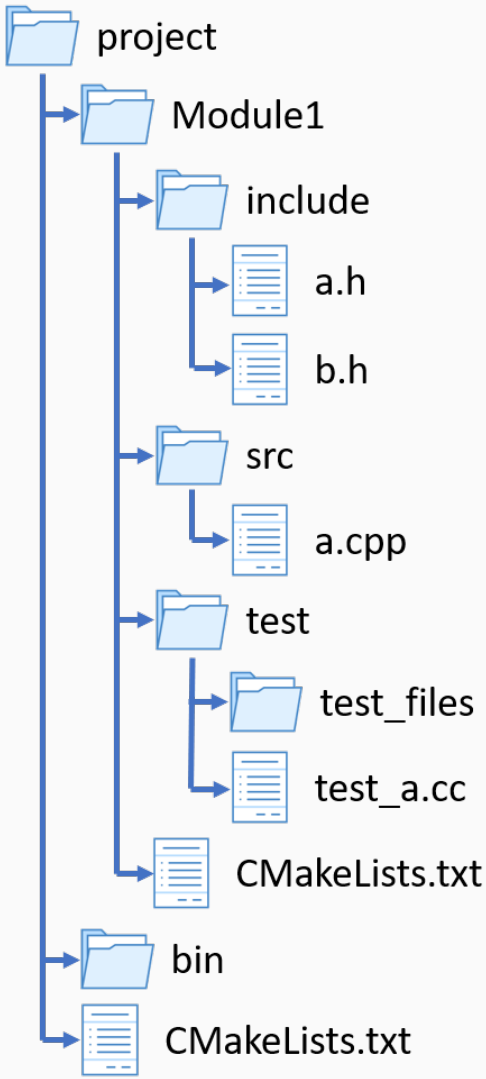
CMake - How can you do it?

include packages

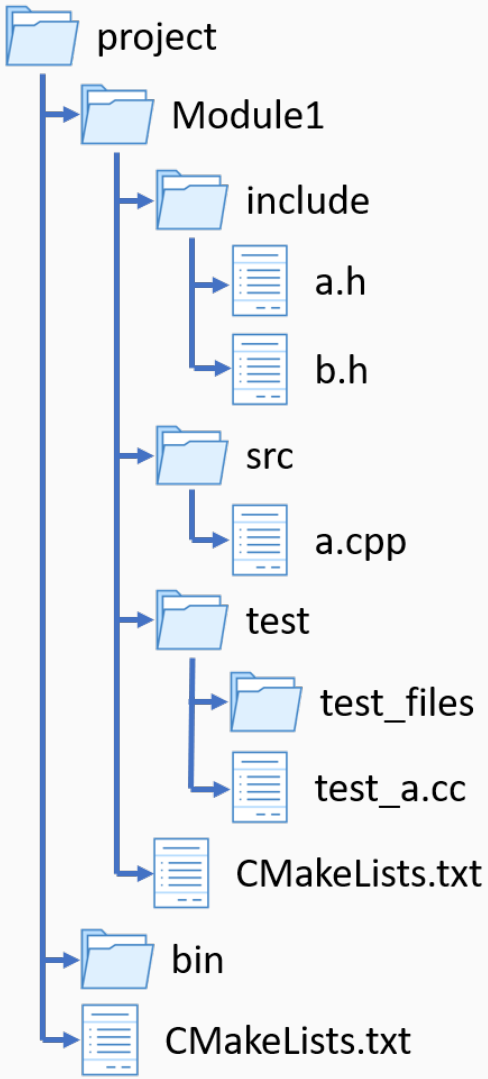
```
find_package( Boost REQUIRED )
# Boost_FOUND, Boost_INCLUDE_DIRS, Boost_LIBRARY_DIRS
# Boost_LIBRARIES, Boost_<COMPONENT>_FOUND, Boost_VERSION
find_package( GTest )
if ( NOT GTEST_FOUND )
endif()
```

The Template

The structure



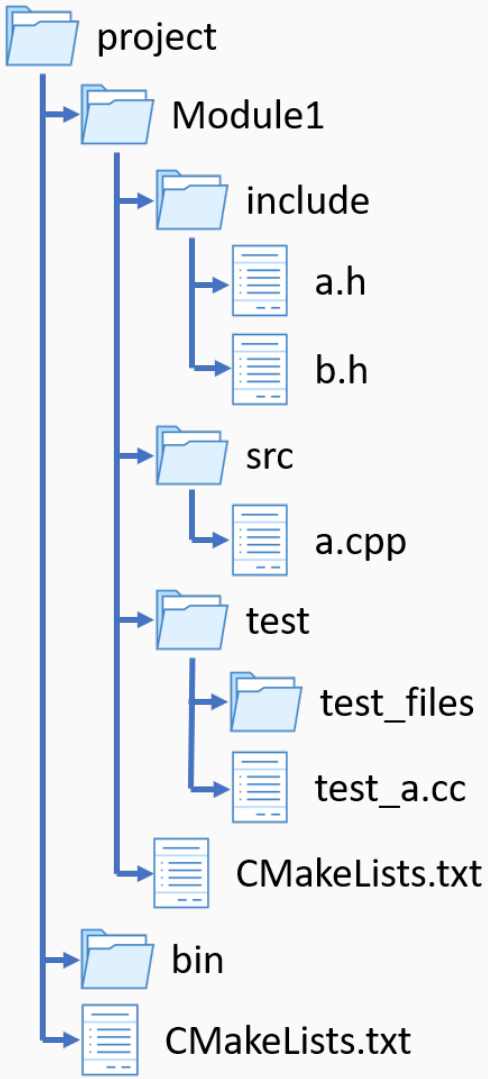
The structure



ToDo for you:

- put all header files into the `include` folder
- put all source files into the `src` folder
- put all test files into the `test` folder
- if you need to link to an other library `libother.so`, include this `target_link_directories(${target} other)`

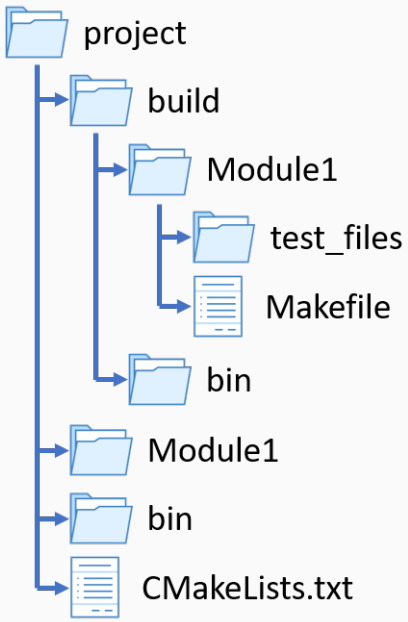
The structure



What is the template doing for you?

- All header files are made available global inside the project
- All header and source files are compiled into a shared library
- For each single test file a executable is created linking at least to the own library

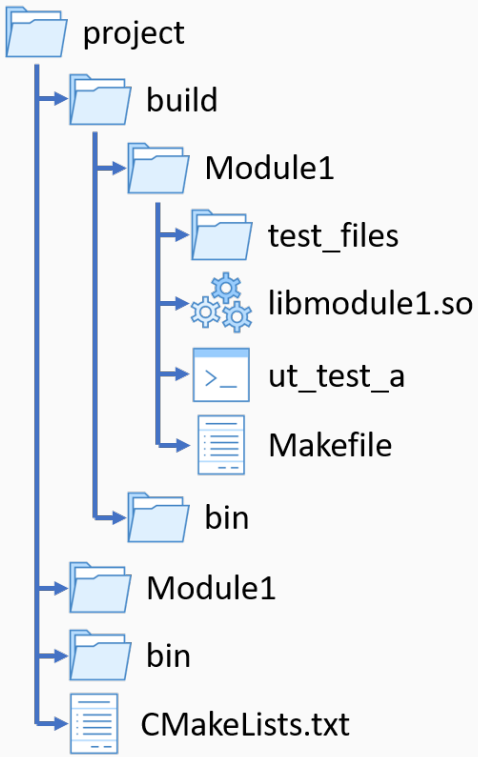
The building



What's happening when calling `cmake`:

- For each module a `Makefile` is created
- For each model the `test_files` folder is copied

The building



What's happening when calling `make`:

- For each module a shared library is created
- For all test files of each model a executable is created

Testing

Testing - What is this good for???

- Correctness
- Easy way to identify errors / bugs
- Quality
- Makes you rethink your software design
- Other people can see in tests, for what the code is written form
- ...

Testing - GTest

What is this?

- Google has created their own test suite for C/C++
- Link to GitHub

Installation:

1. Download → Extract → Switch to folder
2. create a build directory by calling `mkdir build && cd build`
3. to compile and install call
   - `cmake -DBUILD_SHARED_LIBS=ON ..`
   - `make -j2`
   - `sudo make install`

Usage:

- include `#include <gtest/gtest.h>` at the beginning of your code
- add the link flag `-lgtest`

Testing - GTest

Write a test:

- Begin a new test with `TEST(`*basename*`,` *subname*`) { ... }`
- Inside the { ... } write your code
- Test different properties

Test possibilities:

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_EQ(val_1,val_2);	EXPECT_EQ(val_1,val_2);	$val_1 = val_2$
ASSERT_NE(val_1,val_2);	EXPECT_NE(val_1,val_2);	$val_1 \neq val_2$
ASSERT_LT(val_1,val_2);	EXPECT_LT(val_1,val_2);	$val_1 < val_2$
ASSERT_LE(val_1,val_2);	EXPECT_LE(val_1,val_2);	$val_1 \leq val_2$
ASSERT_GT(val_1,val_2);	EXPECT_GT(val_1,val_2);	$val_1 > val_2$
ASSERT_GE(val_1,val_2);	EXPECT_GE(val_1,val_2);	$val_1 \geq val_2$

Testing - GTest

If GTest is not installed on your system, the template will download, compile and link it for you!

GTest has a lot more to offer, check out the links below (or just google it).

Additional helpful links:

- Beginner Tutorial
- Samples
- Advanced Tutorial