

# CMake Tutoriál

Tomáš Milet

Brno University of Technology, Faculty of Information Technology  
Božetěchova 1/2. 602 00 Brno - Královo Pole  
[imilet@fit.vutbr.cz](mailto:imilet@fit.vutbr.cz)

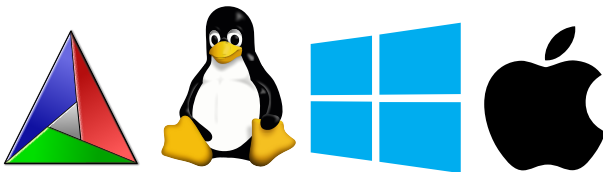


27. října 2018

# Úvod

- <https://cmake.org/cmake/help/v3.13/>
- <https://www.youtube.com/watch?v=bsXLMQ6WgIk>
- <https://pabloariasal.github.io/2018/02/19/its-time-to-do-cmake-right/>

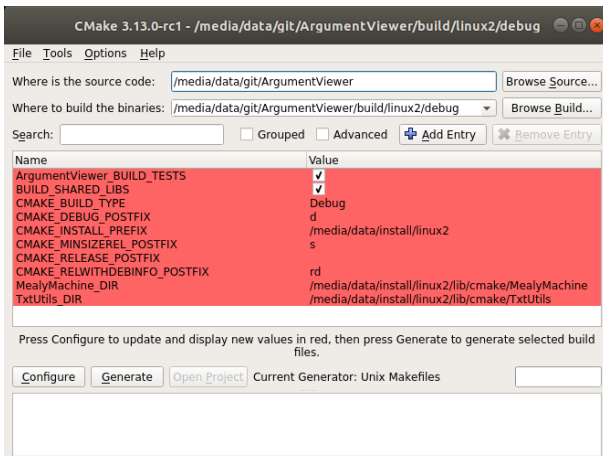
- Multiplatformní
- Usnadňuje práci (závislosti, nastavení, ...)
- Standardizuje sestavování
- Testy, balíčky, uživatelské skripty, ...
- Out of source building, ...



- 1 Odinstalovat všechny starší verze (najít třeba pomocí `$ which cmake`)
- 2 Stáhnout nejnovější binární tar.gr verzi z <https://cmake.org/download/>
- 3 Překopírovat obsah rozbaleného adresáře do `/usr`

```
$ wget https://cmake.org/files/v3.13/cmake-3.13.0-rc1-Linux-x86_64.tar.gz
$ tar xf cmake-3.13.0-rc1-Linux-x86_64.tar.gz
$ cd cmake-3.13.0-rc1-Linux-x86_64
$ sudo cp -r * /usr
$ cmake --version
cmake version 3.13.0-rc1
```

- 1 cmake-gui / ccmake
- 2 Nastavit místo, kde se sestavuje a kde je složka s CMakeLists.txt
- 3 Configure
- 4 Vyřešit závislosti - například nastavit SDL2\_DIR na složku obsahující SDL2Config.cmake
- 5 Generate - vygenerovat makefile / sln file
- 6 make / pustit MSVS



# Základní návod

- Složka obsahující CMakeLists.txt je složka, kterou je možné sestavit
- Hierarchie složek
- CMake sám o sobě nesestavuje aplikace/knihovny
- CMake slouží pro generování makefile / sln souborů
- CMake obsahuje aplikace **cmake cmake-gui ccmake ctest cpack**
- **cmake** - rovnou vygeneruje makefile (parametry můžou ovlivnit generování)
- **cmake-gui** / **ccmake** - uživatelské nastavní parametrů a následné vygenerování makefile
- **ctest** - pro spouštění testů (unit tests, ...)
- **cpack** - pro vytvoření balíčků (\*.deb, ...)



# Prázdný CMake

CMakeLists.txt:

---

```
# minimum version, always use the newest cmake  
cmake_minimum_required(VERSION 3.13.0)
```

---

# HelloWorld CMake

CMakeLists.txt:

---

```
cmake_minimum_required(VERSION 3.13.0)
```

```
# this command will add application HelloWorld to build  
add_executable(HelloWorld main.cpp)
```

---

main.cpp:

---

```
#include<iostream>
```

```
int main(int, char* []) {  
    std::cout << "Hello world!" << std::endl;  
    return 0;  
}
```

---

CMake umožňuje out-of-source building - vytváření \*.o souborů mimo \*.cpp soubory.

```
$ ls
```

```
CMakeLists.txt main.cpp
```

```
$ mkdir build
```

```
$ cd build
```

```
$ cmake ..
```

```
$ ls
```

```
CMakeCache.txt CMakeFiles cmake_install.cmake Makefile
```

```
$ make
```

```
$ ls
```

```
CMakeCache.txt CMakeFiles cmake_install.cmake HelloWorld M
```

# Include Directories

## CMakeLists.txt:

---

```
cmake_minimum_required(VERSION 3.12.2)

add_executable>HelloWorld src/main.cpp include/header.h)

target_include_directories>HelloWorld PUBLIC include)
```

---

## src/main.cpp:

---

```
#include<iostream>
#include<header.h>

int main(int,char*[]){
    std::cout << "header: " << header_function() << std::endl;
    return 0;
}
```

---

## include/header.h:

---

```
#pragma once

inline int header_function(){
    return 10;
}
```

---

# Compile Definitions



## CMakeLists.txt:

---

```
cmake_minimum_required(VERSION 3.12.2)

# this command will add application HelloWorld to build list
add_executable(HelloWorld main.cpp)

target_compile_definitions(HelloWorld PUBLIC PARROT=32)
```

---

## src/main.cpp:

---

```
#include<iostream>

int main(int, char*[]){
    std::cout << "PARROT value: " << PARROT << std::endl;
    return 0;
}
```

---

# Sestavení a instalace knihovny třetí strany

<https://github.com/dormon/Vars>

\$ **tree**

```
.
|
|-- CMakeLists.txt
|-- main.cpp
|-- Vars          # extern library
    |-- CMakeLists.txt
    |-- README.md
    |-- src
        |-- Vars
            |-- Fwd.h
            |-- Resource.cpp
            |-- Resource.h
            |-- Vars.cpp
            |-- Vars.h
            |-- VarsImpl.cpp
            |-- VarsImpl.h
    |-- tests
        |-- catch.hpp
        |-- CMakeLists.txt
        |-- tests.cpp
```

```
$ mkdir varsBuild
$ mkdir install
$ # vygenerování makefile
$ # -H složka s CMakeLists.txt
$ # -B složka pro build
$ # -D nastavení parameteru (proměnné)
$ # CMAKE_INSTALL_PREFIX místo pro instalaci knihovny
$ cmake -HVars -BvarsBuild/ -DCMAKE_INSTALL_PREFIX=install/
$ # spuštění překladu a instalace
$ # --build cesta k build složce
$ # --target název cíle, který se bude sestavovat
$ # pokud cíl závisí na jiných, nejprve se sestaví ty
$ cmake --build varsBuild/ --target install
```

Po instalaci vypadá složka s nainstalovanou knihovnou takto:

```
$ tree install/
|-- include
|   |-- Vars
|       |-- Fwd.h
|       |-- Resource.h
|       |-- vars_export.h
|       |-- Vars.h
|-- lib
    |-- cmake
        |-- Vars
            |-- VarsConfig.cmake
            |-- VarsConfigVersion.cmake
            |-- VarsTargets.cmake
            |-- VarsTargets-noconfig.cmake
    |-- libVars.a
```

# Využití nainstalované knihovny

main.cpp:

---

```
#include<Vars/Vars.h>

int main(int, char* []) {
    vars::Vars vars;
    vars.addBool("AmIBadPerson?", true);
    return 0;
}
```

---

CMakeLists.txt:

---

```
cmake_minimum_required(VERSION 3.13.0)
#search for Vars library
find_package(Vars CONFIG REQUIRED)

add_executable(app main.cpp)
#app needs Vars library for building and linking
#Vars library provides Vars::Vars target
#Vars::Vars target contain every information about
#library (includes, paths to *.so/*.dll *.lib/*.a, defs...
target_link_libraries(app PUBLIC Vars::Vars)
```

```
$ ls
CMakeLists.txt  install  main.cpp  Vars  varsBuild
$ mkdir build
$ # vygenerování makefile pro aplikaci
$ # využívající knihovnu Vars
$ # Vars_DIR je proměnná, která by měla ukazovat na
$ # složku, obsahující soubor VarsConfig.cmake
$ cmake -H. -Bbuild/ -DVars_DIR=install/lib/cmake/Vars
$ # sestavení aplikace
$ cmake --build build/
```



# CMake pro knihovny

- Knihovnu by mělo jít snadno sestavit.
- Knihovna by měla jít nainstalovat.
- Knihovnu by mělo jít sestavit staticy/dynamicky.
- Knihovna by měla propagovat závislosti.
- Knihovna by měla vyřešit obtížné závislosti a nepropagovat problémy dál.
- Knihovna by měla nastavovat správně cesty k \*.lib a \*.h
- Knihovnu by mělo jít snadno vestavět do většího projektu
- Knihovna by neměla předpokládat, že je top most projekt.
- Knihovna by měla mít správně verze.
- Knihovna by měla mít politiku pro kompatibilitu.
- ...

Druhy knihoven:

- dynamické knihovny, sdílené knihovny \*.dll \*.so
- statické knihovny, import knihovny \*.lib \*.a
- header only knihovny \*.h
- object knihovny \*.obj, \*.o
- plugin/module \*.dll \*.so
- ...

Kde se knihovny hledají, když se pustí aplikace:

- Windows - v místě, kde byla aplikace spuštěna, systémové adresáře, adresáře v proměnné PATH
- Linux - zjištění pomocí ldconfig, soubor /etc/ld.so.conf, RPATH, **ne ve working directory!**

Jak zjistit závislosti knihoven/aplikací:

- Windows - dependency walker
- Linux - **ldd lib.so**, jak zjistit RPATH - **objdump -x lib.so | grep -i "runpath"**

Dělení podle druhu kompilace `CMAKE_BUILD_TYPE`:

- RELEASE knihovny
- DEBUG knihovny
- RELWITHDEBINFO
- MINSIZEREL

Dělení podle toho, jestli ji buildíme my nebo ji od někud dostaneme

- IMPORTED - externí knihovna
- SHARED - \*.dll+\*.lib / \*.so
- STATIC - \*.lib / \*.a
- OBJECT - \*.obj / \*.o
- INTERFACE - \*.h
- MODULE - \*.dll / \*.so
- ALIAS - přejmenování

# Statická knihovna

## CMakeLists.txt:

---

```
cmake_minimum_required(VERSION 3.13.0)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE ON)

add_library(engine STATIC engine.h engine.cpp)
```

---

## engine.h:

---

```
#pragma once

int add(int a, int b);
```

---

## engine.cpp:

---

```
#include<engine.h>

int add(int a, int b) {
    return a+b;
}
```

---

# Sdílená knihovna

## CMakeLists.txt:

---

```
cmake_minimum_required(VERSION 3.13.0)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE ON)

add_library(engine SHARED engine.h engine.cpp)

#we need to generated *_export.h file that contains __declspec ...
include(GenerateExportHeader)
generate_export_header(engine EXPORT_FILE_NAME engine_export.h)
```

---

## engine.h:

---

```
#pragma once

#include<engine_export.h>

ENGINE_EXPORT int add(int a, int b);
```

---

## engine.cpp:

---

```
#include<engine.h>

int add(int a, int b) {
    return a+b;
}
```

---



# Header only knihovna

CMakeLists.txt:

---

```
cmake_minimum_required(VERSION 3.13.0)
```

```
add_library(engine INTERFACE)
```

---

engine.h:

---

```
#pragma once
```

```
inline int add(int a, int b) {  
    return a+b;  
}
```

---

```
cmake_minimum_required(VERSION 3.13.0)

project(
  MealyMachine
  VERSION 1.0.2
)

#set these variables to *.cpp, *.c, ..., *.h, *.hpp, ...
set(SOURCES
  src/${PROJECT_NAME}/MealyMachine.cpp
)
set(PRIVATE_INCLUDES )
set(PUBLIC_INCLUDES
  src/${PROJECT_NAME}/Fwd.h
  src/${PROJECT_NAME}/MapTransitionChooser.h
  src/${PROJECT_NAME}/MealyMachine.h
  src/${PROJECT_NAME}/TransitionChooser.h
  src/${PROJECT_NAME}/Exception.h
)
set(INTERFACE_INCLUDES )

#list of libraries to find
#it should contain list of "lists"
#format:
#set(ExternLibraries
#  A\\ B\\ C\\ D
#  E\\ F\\ G
# )
#It will be converted into:
#find_package(A B C D)
#find_package(E F G)
#If version is specified, it has to be the second parameter (B)
set(ExternPrivateLibraries )
set(ExternPublicLibraries )
set(ExternInterfaceLibraries )

#set these variables to targets
set(PrivateTargets )
set(PublicTargets )
set(InterfaceTargets )

#set these libraries to variables that are provided by libraries that does not support configs
set(PrivateIncludeVariables )
set(PublicIncludeVariables )
set(InterfaceIncludeVariables )
set(PrivateReleaseLibraryVariables )
set(PublicReleaseLibraryVariables )
set(InterfaceReleaseLibraryVariables )
set(PrivateDebugLibraryVariables )
set(PublicDebugLibraryVariables )
set(InterfaceDebugLibraryVariables )

SET(CMAKE_CXX_STANDARD 14)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE ON)

option(${PROJECT_NAME}_BUILD_TESTS "toggle building of unit tests")
if(${PROJECT_NAME}_BUILD_TESTS)
  enable_testing()
  add_subdirectory(tests)
  add_test(NAME baseTest COMMAND tests)
endif()
```

---

```

if (!is_publicly_stringlike "")
set(publicly TRUE)
else
set(publicly FALSE)
endif

#find private dependencies
foreach(lib : (getenvPrivateLibraries))
  list_ext lib : (libname)
  if (NOT TARGET : (libname))
    find_package(:lib)
  endif
endforeach

#find public dependencies
foreach(lib : (getenvPublicLibraries))
  list_ext lib : (libname)
  if (NOT TARGET : (libname))
    find_package(:lib)
  endif
endforeach

if (NOT : (isshared))
  option(BUILD_SHARED_LIBS "Build this library as shared")
endif()

set(CMAKE_DEBUG_POSTFIX "d" CACHE STRING "add a postfix, usually d on windows" )
set(CMAKE_RELEASE_POSTFIX "" CACHE STRING "add a postfix, usually empty on windows")
set(CMAKE_RELWITHDEBINFO_POSTFIX "g" CACHE STRING "add a postfix, usually empty on windows")
set(CMAKE_MINGW_RELEASE_POSTFIX "s" CACHE STRING "add a postfix, usually empty on windows")

if (: (isshared))
  add_library(: (PROJECT_NAME) INTERFACE)
else()
  add_library(: (PROJECT_NAME) : (SOURCE) : (PRIVATE_INCLUDES) : (PUBLIC_INCLUDES) : (INTERFACE_INCLUDES))
endif()

add_library(: (PROJECT_NAME) : (PROJECT_NAME) ALIAS : (PROJECT_NAME))

include(BUILD_SHARED_LIBS)

if (: (isshared))
  target_include_directories(: (PROJECT_NAME) INTERFACE : (INSTALL_INTERFACE : (CMAKE_INSTALL_INCLUDEDIR)))
  target_include_directories(: (PROJECT_NAME) INTERFACE : (BUILD_INTERFACE : (CMAKE_CURRENT_SOURCE_DIR) / src))
else()
  target_include_directories(: (PROJECT_NAME) PUBLIC : (INSTALL_INTERFACE : (CMAKE_INSTALL_INCLUDEDIR)))
  target_include_directories(: (PROJECT_NAME) PUBLIC : (BUILD_INTERFACE : (CMAKE_CURRENT_SOURCE_DIR) / src))
endif()

foreach(lib : (PrivateIncludeDirectories))
  target_include_directories(: (PROJECT_NAME) PRIVATE : (lib))
endforeach()

foreach(lib : (PublicIncludeDirectories))
  #this has to be also private because we are exporting this include manually
  target_include_directories(: (PROJECT_NAME) PRIVATE : (lib))
endforeach()

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_INCLUDE_CURRENT_DIR_IN_INTERFACE ON)

if (: (isshared))
  target_link_libraries(: (PROJECT_NAME)
    INTERFACE : (faceFaceTarget)
  )
else()
  target_link_libraries(: (PROJECT_NAME)
    PUBLIC : (PublicTarget)
    PRIVATE : (PrivateTarget)
    INTERFACE : (faceFaceTarget)
  )
endif()

if (: (CMAKE_BUILD_TYPE) STREQUAL "RELEASE")
  foreach(lib : (PrivateDebugLibraryVariables))
    target_link_libraries(: (PROJECT_NAME) PRIVATE : (lib))
  endforeach()
  foreach(lib : (PublicDebugLibraryVariables))
    #this has to be also private, we are exporting manually...
    target_link_libraries(: (PROJECT_NAME) PRIVATE : (lib))
  endforeach()
else()
  foreach(lib : (PrivateDebugLibraryVariables))
    target_link_libraries(: (PROJECT_NAME) PRIVATE : (lib))
  endforeach()
  foreach(lib : (PublicDebugLibraryVariables))
    #this has to be also private, we are exporting manually...
    target_link_libraries(: (PROJECT_NAME) PRIVATE : (lib))
  endforeach()
endif()
endif()

```

---

```
set(PROJECT_NAME_LOWER)
string(TOLOWER ${PROJECT_NAME}) PROJECT_NAME_LOWER

if(PAGE 2 {headerOnly})
    include(HeaderExportInterface)
    generate_export_header(${PROJECT_NAME} EXPORT_FILE_NAME ${PROJECT_NAME}/${PROJECT_NAME_LOWER}_export.h)
    set_target_properties(${PROJECT_NAME} PROPERTY VERSION ${PROJECT_VERSION})
    set_target_properties(${PROJECT_NAME} PROPERTY SOVERSION ${PROJECT_VERSION_MAJOR})
endif()

set_target_properties(${PROJECT_NAME} PROPERTY INTERFACE_${PROJECT_NAME}_MAJOR_VERSION ${PROJECT_VERSION_MAJOR})
set_target_properties(${PROJECT_NAME} PROPERTY PROPERTY_COMPATIBLE_INTERFACE_STRING ${PROJECT_NAME}_MAJOR_VERSION)

install(TARGETS ${PROJECT_NAME} EXPORT ${PROJECT_NAME}Targets
    LIBRARY DESTINATION ${CHUNK_INSTALL_LIBDIR}
    ARCHIVE DESTINATION ${CHUNK_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CHUNK_INSTALL_LIBDIR}
    INCLUDES DESTINATION ${CHUNK_INSTALL_INCLUDEDIR}
)

#install header files
if(PAGE 2 {headerOnly})
    install(
        FILES
            ${CHUNK_CURRENT_RUNTIME_DIR}/${PROJECT_NAME}/${PROJECT_NAME_LOWER}_export.h
        DESTINATION ${CHUNK_INSTALL_INCLUDEDIR}/${PROJECT_NAME}
        COMPONENT Devel
    )
endif()

install(
    FILES
        ${PUBLIC_INCLUDES} ${INTERFACE_INCLUDES}
    DESTINATION ${CHUNK_INSTALL_INCLUDEDIR}/${PROJECT_NAME}
    COMPONENT Devel
)

#source <configuration>.make config file
include(CMakePackageConfigHelpers)
set(CONFIGS ${CHUNK_CURRENT_RUNTIME_DIR}/${PROJECT_NAME})
write_basic_package_version_file(
    ${CONFIGS}/${PROJECT_NAME}.configversion.make
    VERSION ${PROJECT_VERSION}
    COMPATIBILITY SameMajorVersion
)

#source <targets>.make config file
export(EXPORT ${PROJECT_NAME}Targets
    FILE ${CONFIGS}/${PROJECT_NAME}Targets.make
    NAMESPACE ${PROJECT_NAME}::)

#source content of <Config>.make config file
string(CONCAT ConfigContent
    "include(CHUNK独立性macros)\n"
)

foreach(lib ${NativePublicLibraries} ${NativeStaticLibraries})
    list(GET lib 0 libname)
    list(LENGTH lib 1)
    if(1)
        string(LENGTH ConfigContent
            "ConfigContent"
            "lib_dependency" * $libname) "\n"
        )
    else()
        list(GET lib 0 libversion)
        try to match the version
        string(REGEX MATCH "[0-9]+\.[0-9]+\.[0-9]+" matchedVersion ${libversion})
        if("${matchedVersion}" STREQUAL "")
            string(LENGTH ConfigContent
                "ConfigContent"
                "lib_dependency" * $libname) "\n"
            )
        else()
            string(LENGTH ConfigContent
                "ConfigContent"
                "lib_dependency" * $libname * " $libversion) "\n"
            )
        endif()
    endif()
endforeach()

string(LENGTH ConfigContent
    "ConfigContent"
    "include" * "${CHUNK_CURRENT_RUNTIME_DIR}/${PROJECT_NAME}Targets.make)\n"
)

#source <Config>.make config file
file(WRITE ${CONFIGS}/${PROJECT_NAME}.config.make ${ConfigContent})

#install config
set(ConfigPackageLocation lib/make/${PROJECT_NAME})
install(
    FILES
        ${CONFIGS}/${PROJECT_NAME}.config.make
        ${CONFIGS}/${PROJECT_NAME}.configversion.make
    DESTINATION ${ConfigPackageLocation}
    COMPONENT Devel
)

install(EXPORT ${PROJECT_NAME}Targets
    FILE ${PROJECT_NAME}Targets.make
    NAMESPACE ${PROJECT_NAME}::)
install(
    DESTINATION ${ConfigPackageLocation}
)
```

CMake

Špatné knihovny:

- špatné knihovny poskytují pouze proměnné nikoliv TARGET
- špatné knihovny neposkytují instalaci
- špatné knihovny mají špatný install script
- špatné knihovny nemají cmake

Jak řešit?

- Dodat autorům opravený cmake
- Napsat / najít Find\*.cmake script, který vytvoří target za knihovnu
- Distribuovat Find\*.cmake script pro závislosti se svojí knihovnou
- Vytvořit si ručně IMPORTED TARGET
- ...

```
function(getSharedLibraries out lib cfg)
    #message("getSharedLibraries(${lib} ${cfg})")

    if(TARGET ${lib})
        get_target_property(target_type ${lib} TYPE)
        if(${target_type} STREQUAL "INTERFACE_LIBRARY")
            #message("${lib} JE ${target_type}")
            return()
        endif()
    else()
        #message("${lib} NENÍ TARGET")
        return()
    endif()

    get_target_property(dll ${lib} IMPORTED_LOCATION_${cfg})
    list(APPEND dlls ${dll})

    get_target_property(interfaceLibs ${lib} INTERFACE_LINK_LIBRARIES)
    if(NOT "${interfaceLibs}" STREQUAL "interfaceLibs-NOTFOUND")
        foreach(interfaceLib ${interfaceLibs})
            getSharedLibraries(ilibs ${interfaceLib} ${cfg})
            list(APPEND dlls ${ilibs})
        endforeach()
    endif()
    list(REMOVE_DUPLICATES dlls)
    set(${out} ${dlls} PARENT_SCOPE)
endfunction()

function(getAllSharedLibraries allLibraries app cfg)
    get_target_property(libs ${app} LINK_LIBRARIES)
    foreach(lib ${libs})
        getSharedLibraries(libList ${lib} ${cfg})
        #message("${lib} ##### ${libList}")
        list(APPEND allLibs ${libList})
    endforeach()
    list(REMOVE_DUPLICATES allLibs)
    set(${allLibraries} ${allLibs} PARENT_SCOPE)
endfunction()

getSharedLibraries(allDebugSharedLibraries ${PROJECT_NAME} DEBUG)
getSharedLibraries(allReleaseSharedLibraries ${PROJECT_NAME} RELEASE)

#message("SharedDebug : ${allDebugSharedLibraries}")
#message("SharedRelease: ${allReleaseSharedLibraries}")

if(MSVC)
    foreach(lib ${allDebugSharedLibraries})
        file(COPY ${lib} DESTINATION ${CMAKE_CURRENT_BINARY_DIR}/Debug)
    endforeach()
    foreach(lib ${allReleaseSharedLibraries})
        file(COPY ${lib} DESTINATION ${CMAKE_CURRENT_BINARY_DIR}/Release)
    endforeach()
endif()

install(TARGETS ${PROJECT_NAME} RUNTIME DESTINATION .)
install(FILES ${allDebugSharedLibraries} DESTINATION .)
```



---

```
cmake_minimum_required(VERSION 3.13.0)

include(FetchContent)

FetchContent_Declare(MealyMachine      GIT_REPOSITORY https://github.com/dormon/MealyMachine.git   )
FetchContent_Declare(TxtUtils          GIT_REPOSITORY https://github.com/dormon/TxtUtils.git       )
FetchContent_Declare(ArgumentViewer    GIT_REPOSITORY https://github.com/dormon/ArgumentViewer.git  )

FetchContent_GetProperties(MealyMachine)
if(NOT MealyMachine_POPULATED)
    FetchContent_Populate(MealyMachine)
    add_subdirectory(${mealmachine_SOURCE_DIR} ${mealmachine_BINARY_DIR})
endif()

FetchContent_GetProperties(TxtUtils)
if(NOT TxtUtils_POPULATED)
    FetchContent_Populate(TxtUtils)
    add_subdirectory(${txtutils_SOURCE_DIR} ${txtutils_BINARY_DIR})
endif()

FetchContent_GetProperties(ArgumentViewer)
if(NOT ArgumentViewer_POPULATED)
    FetchContent_Populate(ArgumentViewer)
    add_subdirectory(${argumentviewer_SOURCE_DIR} ${argumentviewer_BINARY_DIR})
endif()

add_executable(as main.cpp)
target_link_libraries(as ArgumentViewer::ArgumentViewer)
```

---

Thank you for your attention! Questions?