Modern CMake

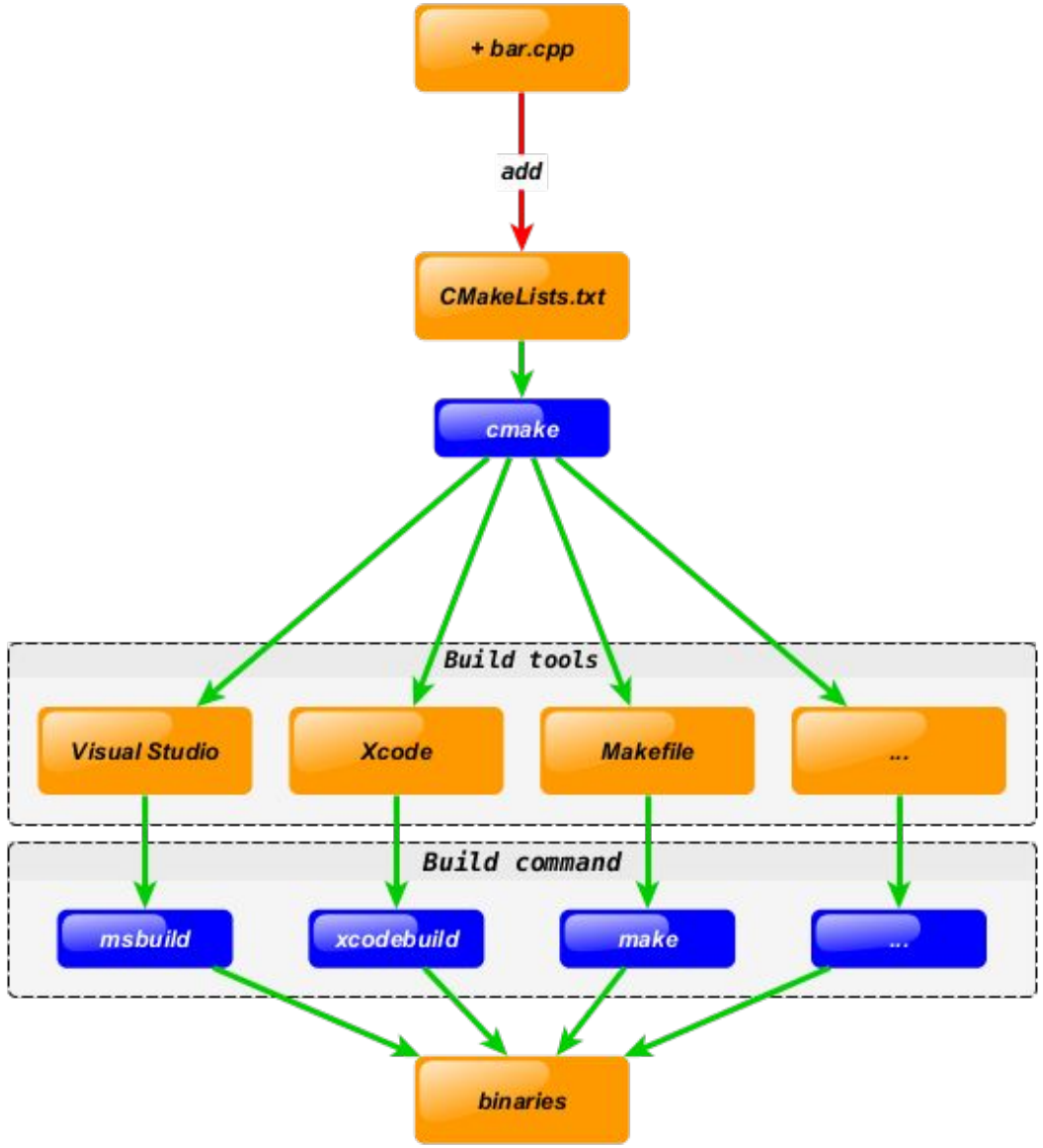
Dariusz Grabowski

Czym jest cmake?

CMake

- multiplatformowe narzędzie do automatycznego zarządzania procesem kompilacji (w tym cross kompilacji)
- generuje pliki opisujące proces kompilacji dla różnych środowisk i systemów (np. Makefile, Ninja, Eclipse, SublimeText, MS Visual Studio i wiele innych)
- umożliwia tworzenie paczek instalacyjnych (CPack)
- umożliwia wykonywanie testów komponentu (CTest)

Cmake


+ bar.cpp
add
CMakeLists.txt
cmake
Build tools
Visual Studio
Xcode
Makefile
...
Build command
msbuild
xcodebuild
make
...
binaries

Przykład CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15)

project("SomeTemplateProject")

add_subdirectory(software/gs)
add_subdirectory(software/ut)

enable_testing()
include(CTest)
add_test(NAME ATest COMMAND ./ATest)

set(CPACK_PACKAGE_NAME "MyExampleProject")
set(CPACK_PACKAGE_VERSION "1.0.0")
set(CPACK_GENERATOR "TGZ")

include(CPack)
```

Tworzenie targetów

```
# ------ binaria

add_executable(exampleMain
    src/exampleMain.cpp
)
add_executable(TemplateProjNamespace::exampleMain ALIAS exampleMain)


# ------ "bilioteki"

add_library(exampleClassLib
    src/SomeClassExample.cpp
)

add_library(TemplateProjNamespace::exampleClassLib ALIAS exampleClassLib)
```

Tworzenie targetów

```
add_library(exampleClassLib SHARED
    src/SomeClassExample.cpp
)

# ------ możliwe rodzaje:
# SHARED - linkowane dynamicznie w runtime (*.so)
# STATIC - skompilowane archiwum używane w trakcie linkowania targetu (*.a)
# OBJECT - skompilowane pliki źródłowe możliwe do wykorzystania przy
tworzeniu innego targetu
# INTERFACE - wirtualny target - nie generuje żadnego outputu
# ALIAS - tworzy alias do innego targetu


# UWAGA: zamiast SHARED|STATIC użyj:
# cmake -DBUILD_SHARED_LIBS ..
```

Tworzenie targetów

```
# --- UŻYWANIE OBJECT LIBRARY

add_library(exampleLib OBJECT
    SomeSharedSource.cpp
)

# --- sposób pierwszy:
add_library(target1 SHARED
    target1.cpp
)
target_link_libraries(target1 PUBLIC exampleLib)

# --- sposób drugi:
add_executable(target2
    target2.cpp
    $<TARGET_OBJECTS:exampleLib>
)
```

Tworzenie targetów

```
# --- UŻYWANIE INTERFACE LIBRARY


add_library(common INTERFACE)

set_target_properties(common PROPERTIES
        PUBLIC_HEADER "ReceivingInterface.h;SendingInterface.h")

target_compile_definitions(common INTERFACE -DUSE_SOME_EXTRAS)
```

Właściwości targetów

```
# poziomy widoczności target properties:
#   PRIVATE - tylko dla targetu
#   PUBLIC - dla targetu i jego użytkowników
#   INTERFACE - tylko dla użytkowników

# ------------ DOSTĘPNE POLECENIA:
# ustawianie linkowania
target_link_libraries(someLib PUBLIC other)

# ustawianie katalogów z plikami nagłówkowymi
target_include_directories(someLib PUBLIC someLib/include)

# dodawanie flag kompilatora do targetu
target_compile_options(someLib PUBLIC -Wall)

# dodawanie definicji do targetu
target_compile_definitions(someLib PRIVATE -DFOO)

# dodawanie zdolności kompilatora do targetu
target_compile_features(someLib PRIVATE c_std_11)
```

Generowanie plików

```
// SomeHeaderTemplate.h.in

#pragma once

#define SOME_GENERATED_INFO "@SOME_CMAKE_VALUE@"
```

```
# CMakeLists.txt

set(SOME_CMAKE_VALUE "Foo")

configure_file(
    SomeHeaderTemplate.h.in
    somePath/SomeHeaderTemplate.h
)
```

```
// SomeHeaderTemplate.h

#pragma once

#define SOME_GENERATED_INFO "Foo"
```

Zmienne CMake

- CMAKE_VERSION
- PROJECT_NAME
- CMAKE_CURRENT_SOURCE_DIR
- CMAKE_CTEST_COMMAND
- CMAKE_CURRENT_BINARY_DIR
- PROJECT_VERSION
- CMAKE_TOOLCHAIN_FILE
- CTEST_MEMORYCHECK_COMMAND
- CTEST_SVN_COMMAND
- . . .
-
- https://cmake.org/cmake/help/latest/manual/cmake-variables.7.html

Eksport targetów

```
add_library(myLib MyLib.cpp)

# Można podać 1 lub więcej targetów
install(TARGETS myLib EXPORT myLibTargets
        RUNTIME DESTINATION bin # binaria
        LIBRARY DESTINATION lib # *.so
        ARCHIVE DESTINATION lib # *.a
        PUBLIC_HEADER DESTINATION include # pliki nagłówkowe
)

# miejsce gdzie ma zostać zapisany plik modułu
install(EXPORT myLibTargets DESTINATION lib/cmake/myLib NAMESPACE myLib::)


include(CPack)
```

Import targetów

```
# opcjonalnie REQUIRED
find_package(Boost 1.65 COMPONENTS filesystem regex)

add_library(myLib MyLib.cpp)

if(Boost_FOUND)
     target_link_libraries(myLib PUBLIC Boost::regex)
endif()
```

Dodawanie testów

```
add_executable(testBinary Test.cpp)

add_test(NAME MyTest COMMAND ./testBinary [args] WORKING_DIRECTORY ./path)

# ------------------------

include(GoogleTest)
gtest_discover_tests(testBinary WORKING_DIRECTORY ./path)
```

Toolchain

```
SET(CMAKE_SYSTEM_NAME Linux)
SET(CMAKE_SYSTEM_VERSION 1)
SET(CMAKE_SYSTEM_PROCESSOR arm)

# specyfikacja kompilatora
SET(CMAKE_C_COMPILER   /opt/gcc-linaro-5.4.1-2017/bin/arm-linux-gnueabihf-gcc)
SET(CMAKE_CXX_COMPILER /opt/gcc-linaro-5.4.1-2017/bin/arm-linux-gnueabihf-g++)

# alternatywny root dla szukania paczek o modułów
SET(CMAKE_FIND_ROOT_PATH
${CMAKE_FIND_ROOT_PATH};/opt/gcc-linaro-5.4.1-2017/arm-linux-gnueabihf/libc)

# czy mogą być używane biblioteki i paczki z systemu
SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
# szukaj tylko w podkatalogu cross compilatora
SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)


# dodajemy przy wywołaniu cmake: cmake -DCMAKE_TOOLCHAIN_FILE=/path/to/Toolchain.cmake ..
```

Dobre praktyki

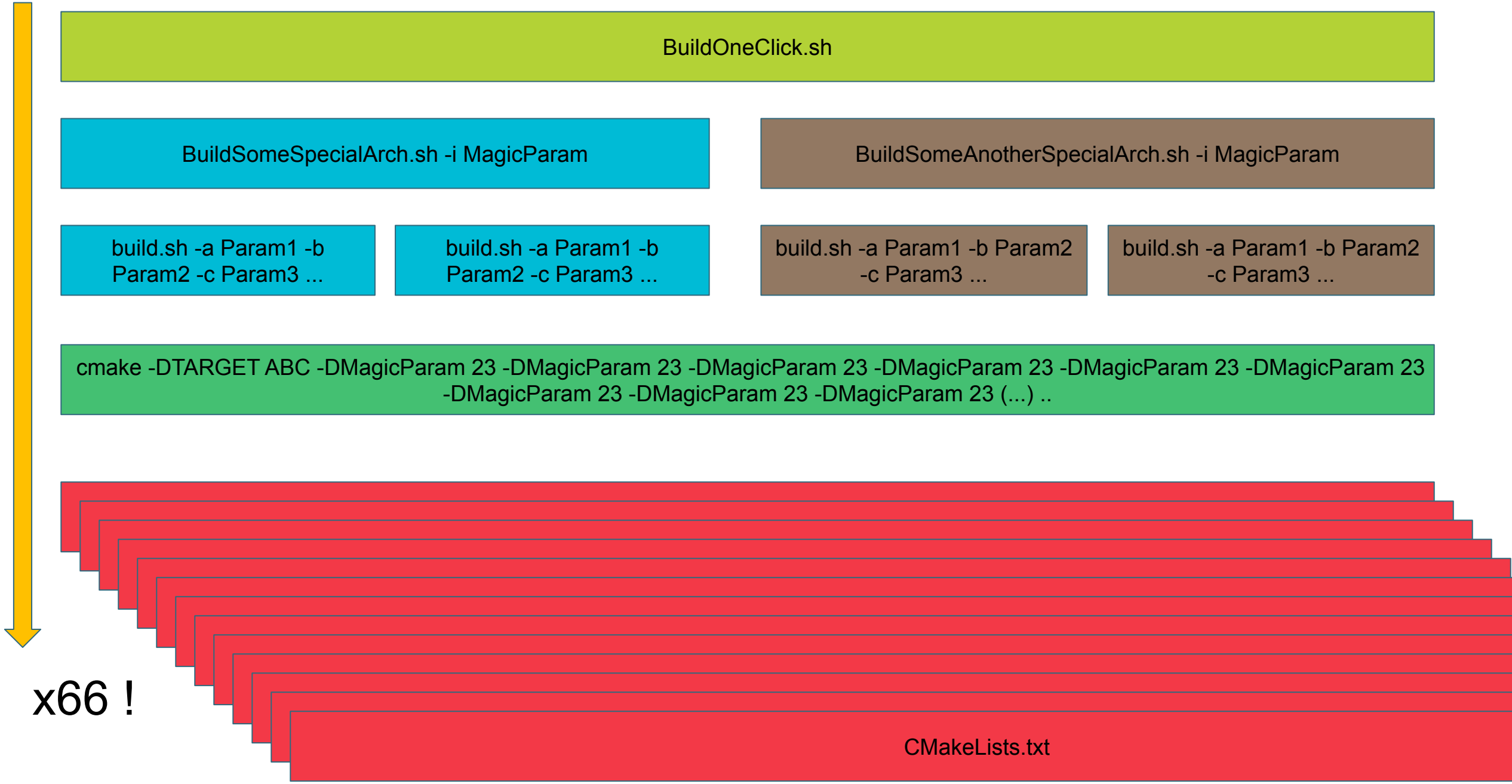
- dziel projekty na małe kawałki (biblioteki lub przynajmniej podsystemy)
- jeśli potrzebujesz już generować jakieś pliki użyj mechanizmów cmake’a
- nie używaj własnych funkcji
- nigdy nie modyfikuj CMAKE_CXX_FLAGS : SET (CMAKE_CXX_FLAGS "-std=c++11")
- nie używaj add_definitions() include_directories() itd.
- unikaj file(GLOB)
- nie przykrywaj cmake skryptami!

Alternatywa dla file(GLOB)

```
# zadziała tylko przy pierwszym uruchomieniu cmake
file(GLOB SRC_FILES src/*.cpp)

add_library(myTestLib ${SRC_FILES})


# dużo lepiej:
add_executable(myTestLib
    src/ClassA.cpp
    src/ClassB.cpp
)
```


BuildOneClick.sh
BuildSomeSpecialArch.sh -i MagicParam
BuildSomeAnotherSpecialArch.sh -i MagicParam
build.sh -a Param1 -b Param2 -c Param3 ...
build.sh -a Param1 -b Param2 -c Param3 ...
build.sh -a Param1 -b Param2 -c Param3 ...
build.sh -a Param1 -b Param2 -c Param3 ...
cmake -DTARGET ABC -DMagicParam 23 -DMagicParam 23 -DMagicParam 23 -DMagicParam 23 -DMagicParam 23 -DMagicParam 23 -DMagicParam 23 -DMagicParam 23 -DMagicParam 23 (...) ..
x66 !
CMakeLists.txt

Pytania?

Zadanie

1. Pobrać źródła projektu:

   `git clone https://github.com/grabowski-d/ci-training-cmake.git`
2. Zapoznać się ze źródłami i zależnościami, przejrzeć plik CMakeLists.txt
3. Zakomentować wskazaną sekcję w pliku CMakeLists.txt, odkomentować testową
4. Doprowadzić do budowania się projektu

Wymagania/ograniczenia:

1. Dla każdego podkatalogu powinien być stworzony plik CMakeLists.txt
2. Zabronione jest używanie poleceń nie związanych z targetami

   (include_directories() add_definitions() itd.)
3. Zabroniona jest modyfikacja plików źródłowych
4. Po udanym zbudowaniu należy uruchomić testy (poprzez polecenie cmake).

   Wszystkie testy powinny przechodzić poprawnie.