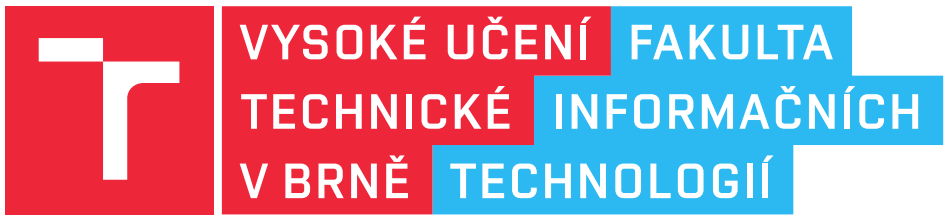
Meta sestavovací systémy

Viktor Malík

Fakulta informačních technologií Vysokého učení technického v Brně,
Božetěchova 1/2, 612 66 Brno – Královo Pole

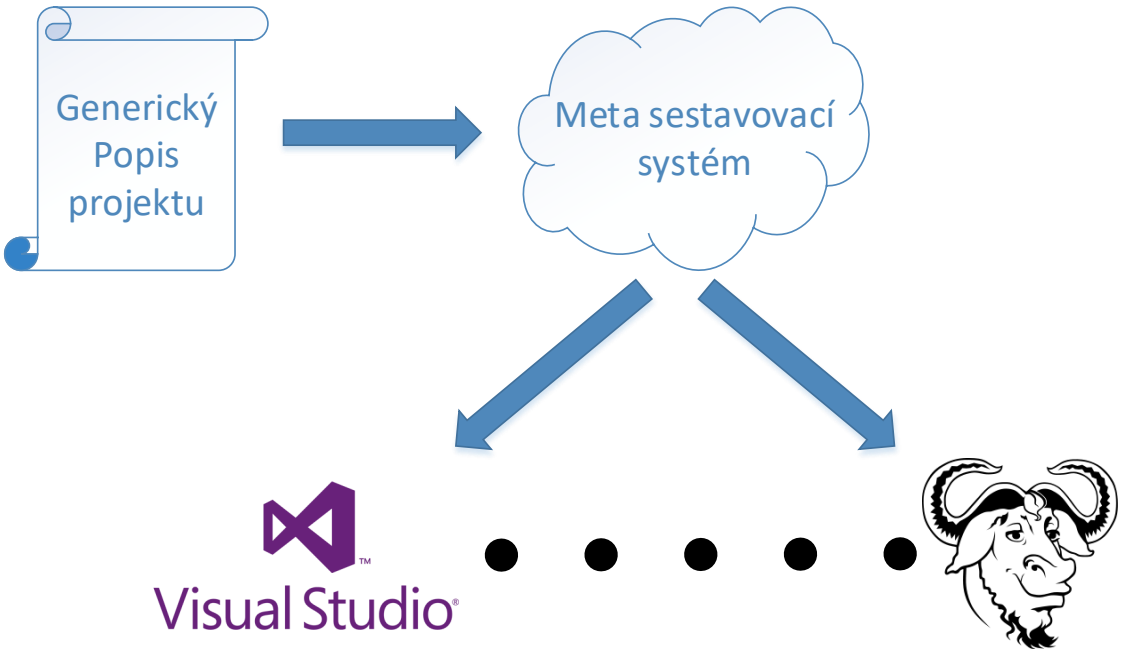
imalik@fit.vutbr.cz



13. března 2024

Meta sestavovací systémy

- Meta sestavovací systémy
  - CMake, QMake, Meson, GYP, Google Blueprint (Go Lang)
- Na základě univerzálního popisu generují konkrétní popis pro specifický sestavovací systém, platformu a prostředí
  - Visual Studio, Make, Ninja, ...



Požadavky na meta sestavovací systém

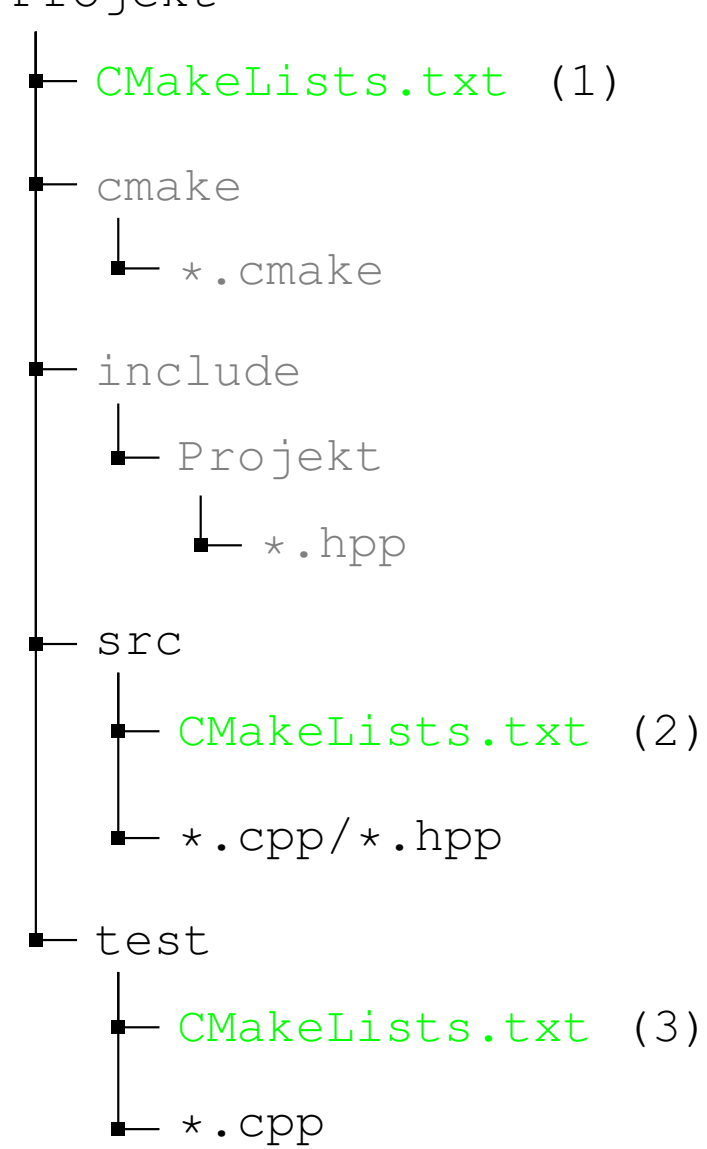
- Přenositelnost mezi platformami a prostředími
  - Systém musí být dostupný na co největším počtu platforem (např. Linux, Windows, ...)
  - Systém musí být schopen nalézt potřebné závislosti v daném prostředí (např. různé umístění knihoven)
- Podpora různých sestavovacích systémů a překladačů
  - Make, Ninja, Visual Studio, ...
  - GCC, Intel, MSVC, ...
- Snadná rozšiřitelnost a modifikovatelnost
  - pro nové prostředí a nástroje

- Primární zaměření na projekty v C/C++
- Multiplatformní s širokou podporou sestavovacích systémů
- Vyhledávání závislostí
  - Vyhledávání závislostí v systémových cestách
  - Získávání závislostí (z repozitářů, tar balíčků atd.)
- Podpora překladu mimo zdrojový kód (out of tree build)
- Modulární návrh, snadná rozšiřitelnost
- Integrace nástrojů jako tar/gzip (pro Windows)
- Základní podpora pro generování kódu
- Vestavěné nástroje pro testování (CTest) a vytváření balíčků (CPack)

- CMakeLists.txt ve zdrojovém stromu
  - Reprezentuje adresář projektu
  - V kořenovém adresáři reprezentuje vstupní bod CMake
  - Obsahují specifikaci sestavení
- Soubory se skripty CMake (*.cmake)
  - Lze vykonat pomocí „`> cmake -P <skript>`“
  - Skripty nesmí definovat cíle a provádět akce
- Soubory modulů CMake (*.cmake)
  - Lze je vkládat do „CMakeLists.txt“ a skriptů
  - Systémové moduly obvykle v „/usr/share/cmake/Modules/“
  - Prohledávány jsou také cesty v CMAKE_PREFIX_PATH

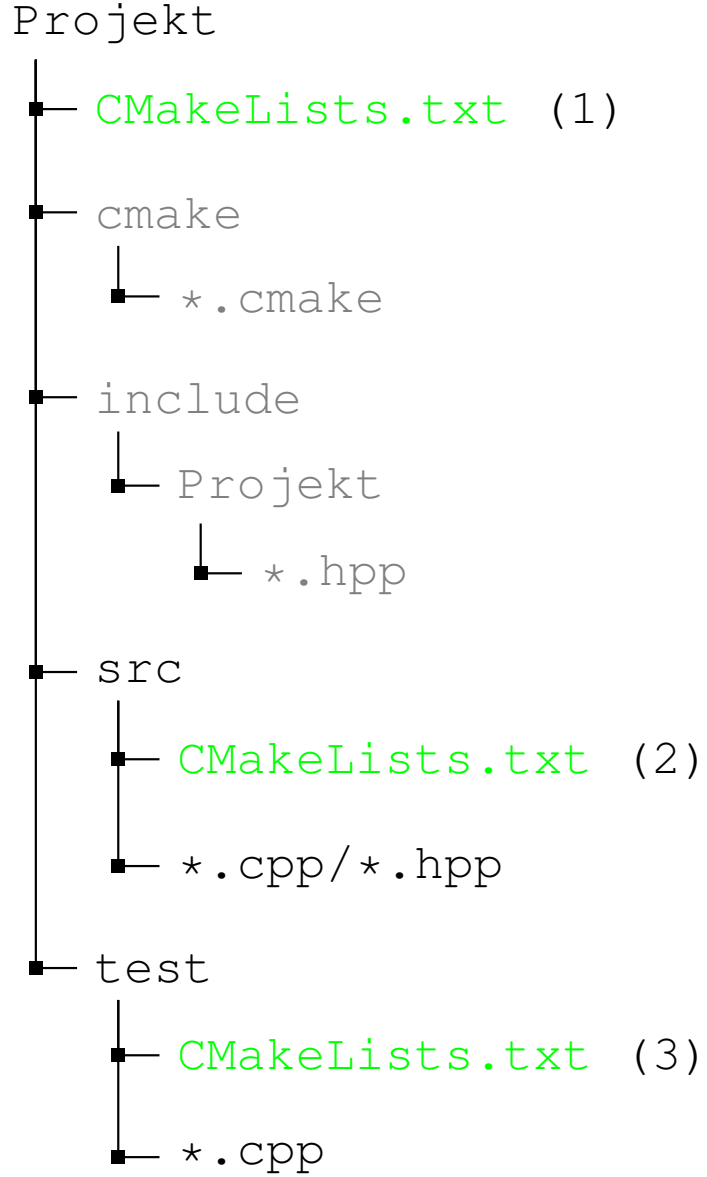
CMake – Typická struktura projektu

- CMakeLists.txt
  1. Konfigurace závislostí, platformy pro celý projekt
  2. Definuje hlavní cíle
  3. Konfigurace testů



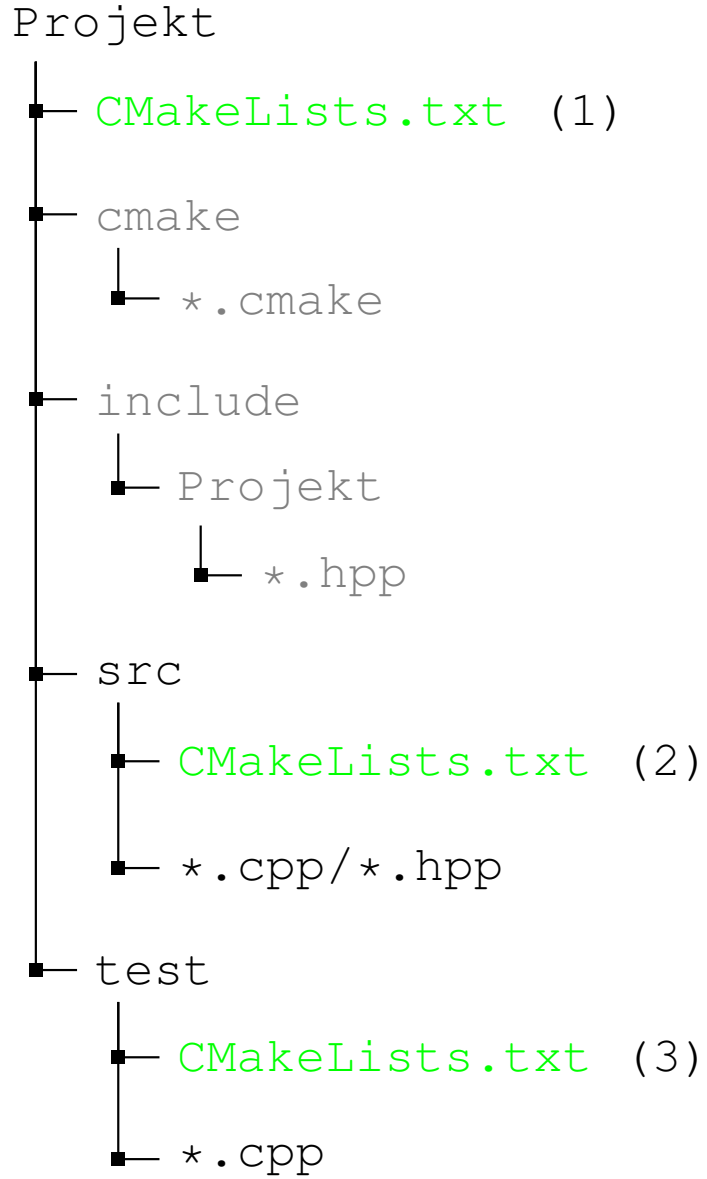
CMake – Typická struktura projektu

- CMakeLists.txt
  1. Konfigurace závislostí, platformy pro celý projekt
  2. Definuje hlavní cíle
  3. Konfigurace testů
- Adresář „cmake“ lokální moduly CMake



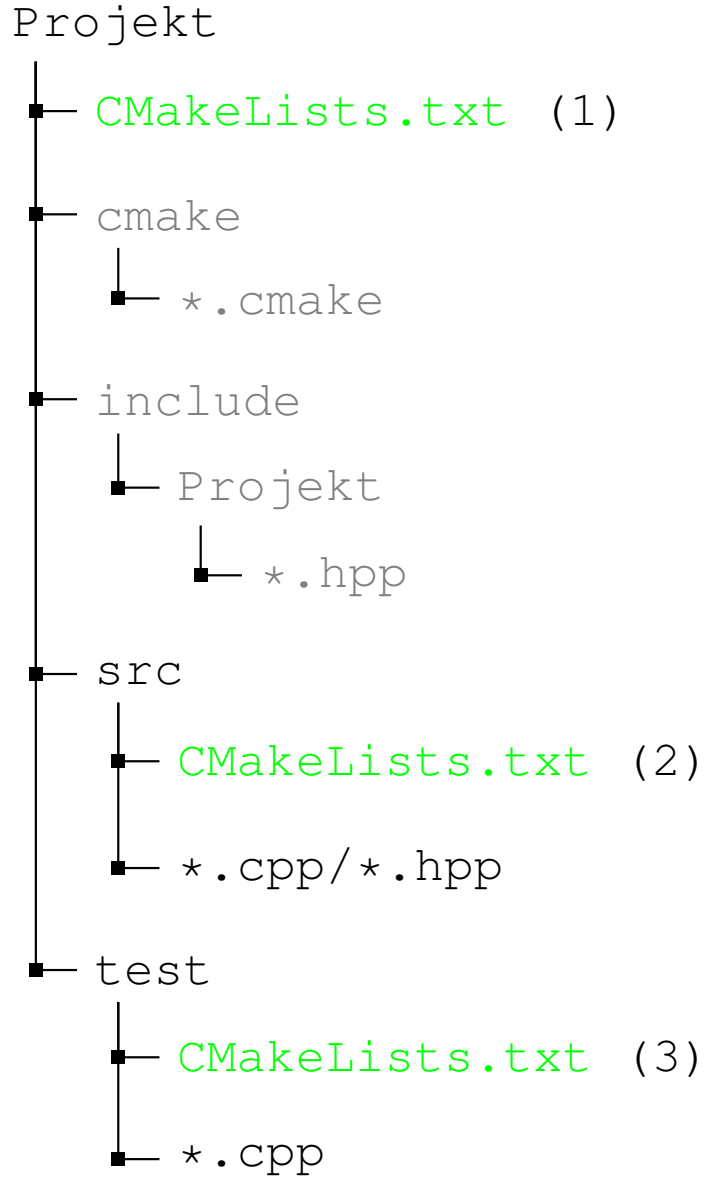
CMake – Typická struktura projektu

- CMakeLists.txt
  1. Konfigurace závislostí, platformy pro celý projekt
  2. Definuje hlavní cíle
  3. Konfigurace testů
- Adresář „cmake“ lokální moduly CMake
- Adresář „include“ je důležitý především pro knihovny



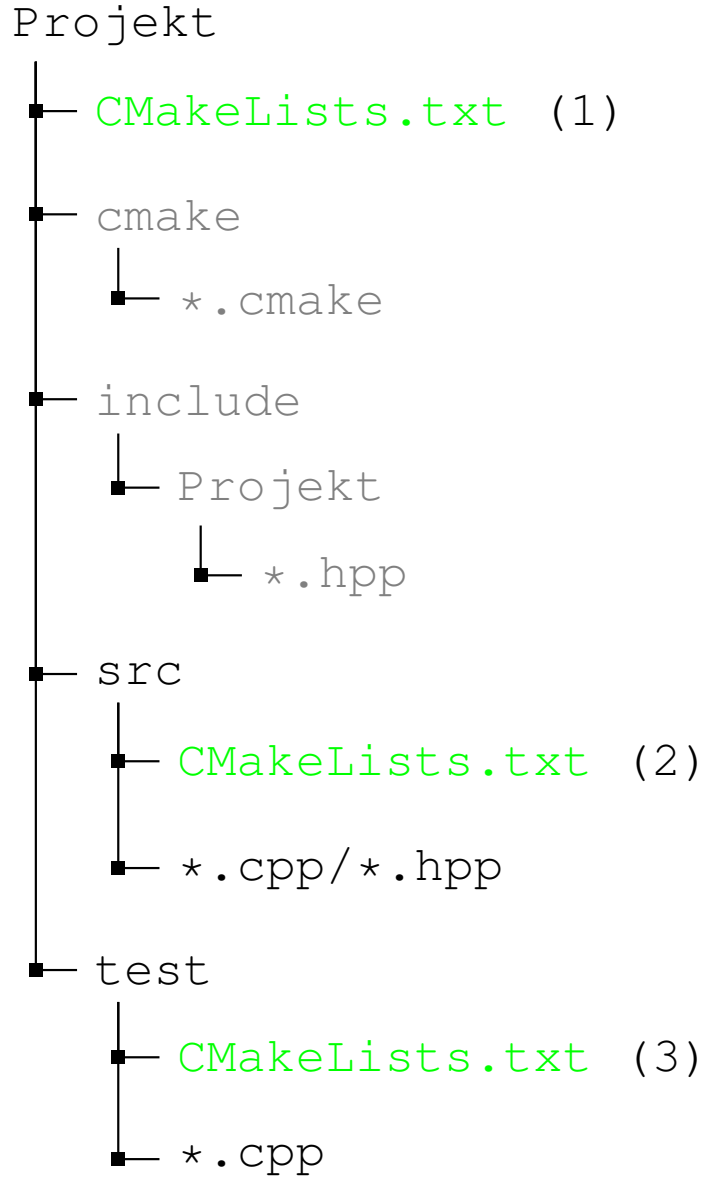
CMake – Typická struktura projektu

- CMakeLists.txt
  1. Konfigurace závislostí, platformy pro celý projekt
  2. Definuje hlavní cíle
  3. Konfigurace testů
- Adresář „cmake“ lokální moduly CMake
- Adresář „include“ je důležitý především pro knihovny
- Adresář „src“ obsahuje hlavní zdrojové kódy projektu



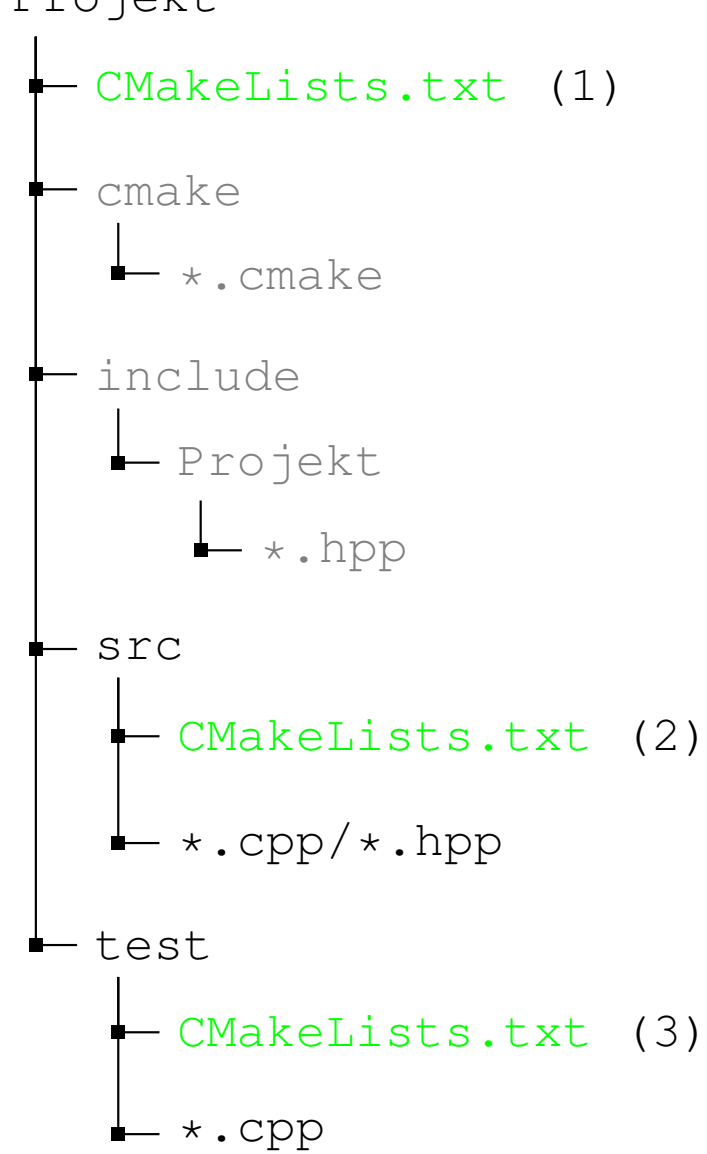
CMake – Typická struktura projektu

- CMakeLists.txt
  1. Konfigurace závislostí, platformy pro celý projekt
  2. Definuje hlavní cíle
  3. Konfigurace testů
- Adresář „cmake“ lokální moduly CMake
- Adresář „include“ je důležitý především pro knihovny
- Adresář „src“ obsahuje hlavní zdrojové kódy projektu
- Adresář „test“ obsahuje definice testů a testovací data



CMake – Typická struktura projektu

- CMakeLists.txt
  1. Konfigurace závislostí, platformy pro celý projekt
  2. Definuje hlavní cíle
  3. Konfigurace testů
- Chybí adresář „doc“ s cílem pro generování dokumentace



- `cmake [parametry] <Cesta k CMakeLists.txt>`
  - Generování sestavovacího systému (s výchozím nastavením pro aktuální platformu)

    ```
    > cmake [cesta-ke-zdroji]
    ```

  - Výběr generátoru (např. Unix Makefiles – dle platformy)

    ```
    > cmake -G <nazev-generatoru> [cesta-ke-zdroji]
    ```

  - Definice proměnné prostředí CMake

    ```
    > cmake -D <nazev>=<hodnota> ... [cesta-ke-zdroji]
    ```

  - Spuštění překladu vygenerovaného projektu

    ```
    > cmake --build <cesta-k-projektu> [--target <cil>]
    ```

- Chybí „simulované vykonávání" – pro meta systém nemá význam

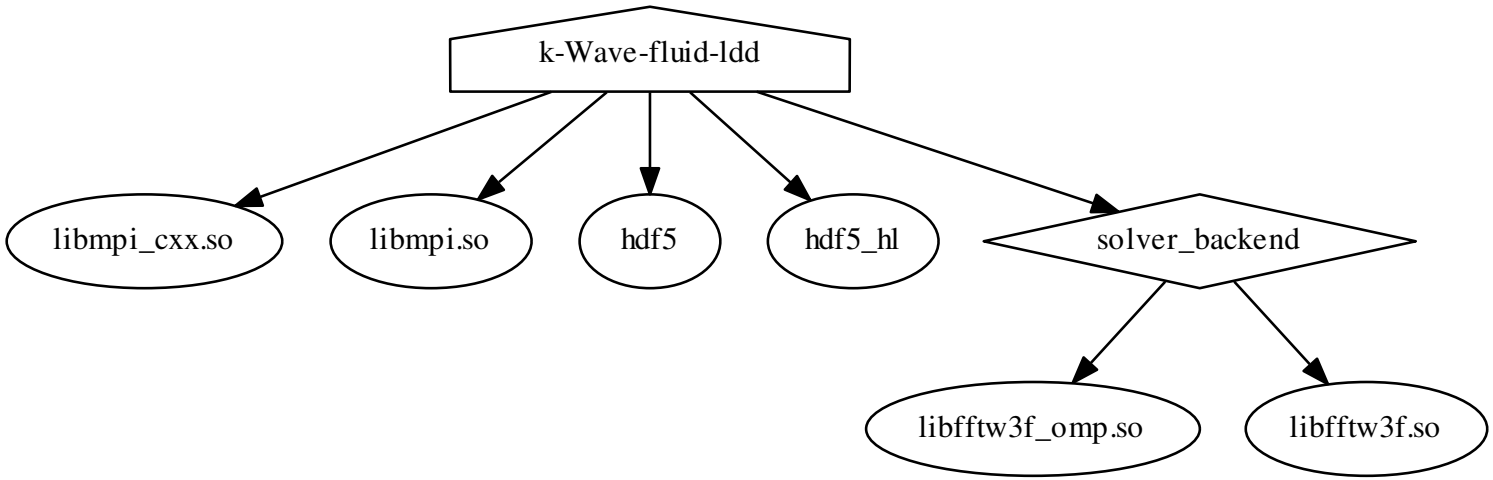
```
> cmake --trace [cesta-ke-zdroji]
> cmake --trace-source=<soubor> [cesta-ke-zdroji]
```

- Ladění CMake skriptů

```
> cmake --debug-output [cesta-ke-zdroji]
```

- Log soubory:
  `CMakeFiles/CMakeOutput.log`
  `CMakeFiles/CMakeError.log`
- Generování grafu závislostí

```
> cmake --graphviz=[soubor] [cesta-ke-zdroji]
```



CMakeLists.txt – základy syntaxe

- Posloupnost příkazů oddělených odřádkováním
  - `<identifikátor>([argumenty])`
  - Argumenty mohou být odděleny pomocí mezer nebo odřádkováním
  - `add_executable(hello world.c)`
- CMake je case-insensitive!
- Téměř všechny programové struktury (včetně řídících) jsou příkazy v tomto formátu!

```
set(MY_VAR "Hello World")

function(MY_FUNC arg)
# Kód funkce
endfunction(MY_FUNC)

include(my_module)
```

```
if(NOT DEFINED Foo)
    message("Foo undefined!")
endif()

foreach(arg "Hello" "World")
    message("${arg}")
endforeach()
```

CMakeLists.txt – proměnné

- Nastavovány pomocí `set(MY_VAR hodnota)`
- Platné v aktuálním a pod-adresářích (nebo funkcích)
- Hodnota proměnné v nadřazeném adresáři se neaktualizuje!
  - Lze změnit pomocí `set(MY_VAR hodnota PARENT_SCOPE)`
- K proměnné lze přistupovat jako

```
message(${MY_VAR}) # Proměnná CMake
message($ENV{ENV_VAR}) # Proměnná prostředí
```

- Uživatelské dvoustavové proměnné
  - `option(MY_VAR "popis proměnné" [výchozí-hodnota])`
  - Mohou nabývat hodnot `ON` a `OFF`
  - Lze zobrazit pomocí

```
> cmake -LAH
> cmake-gui
```

- Znovupoužitelnost částí kódu je umožněna pomocí funkcí a maker
- Návratové hodnoty funkcí řešeny úpravou rodičovských proměnných
- Makra
  - Pracují ve stejném rámci proměnných jako „volající“
  - Parametry pouze textovou náhradou, nejsou opravdovými proměnnými
- Funkce
  - Vytvářejí nový rámec – úpravy proměnných jsou pouze lokální (vyžadují použití `PARENT_SCOPE`)
  - Parametry jsou proměnnými

```
function(TEST arg1 ret)
  # ...
  set(${ret} ${arg1} PARENT_SCOPE)
endfunction(TEST)
```

```
macro(TEST_MACRO arg1 ret)
  # ...
  set(${ret} ${arg1})
end(TEST_MACRO)
```

- Soubor Projekt/CMakeLists.txt

```
# Minimální verze CMake
cmake_minimum_required(VERSION 3.8)
project(název-projektu) # Název projektu

# Rekurzivně prohledá adresáře projektu
# a vybere *.h/*.hpp soubory
file(GLOB_RECURSE HEADERS "*.h" "*.hpp")

set(SOURCES main.cpp) # Zdrojové soubory
add_subdirectory(src) # Kód projektu
add_subdirectory(test) # Definice testů

set(CMAKE_CXX_STANDARD 11) # --std=c++11
# Spustitelný cíl s názvem "název-projektu"
add_executable(${PROJECT_NAME}
               ${SOURCES} ${HEADERS})
target_link_libraries(${PROJECT_NAME} hdf5)
```

CMake – typická struktura

- Soubor Projekt/src/CMakeLists.txt

```
# Přidání zdrojových souborů do proměnné
# nadřazeného skriptu
set(SOURCES ${SOURCES}
    ${CMAKE_CURRENT_SOURCE_DIR}/foo.cpp
    ${CMAKE_CURRENT_SOURCE_DIR}/bar.cpp
    PARENT_SCOPE)
```

- Soubor Projekt/test/CMakeLists.txt

```
# Cíl pro testování pomocí CTest
enable_testing()
add_test(TestA ${PROJECT_NAME})
add_test(TestB ${PROJECT_NAME})
```

CMake – definice cílů

- Binární cíle
- `add_executable(<název> source1 [source2 ...])`
  - Cíl pro překlad spustitelného souboru ze zdrojových nebo objektových souborů
- `add_library(<název> source1 [source2 ...]))`
- `add_library(<název> OBJECT <sources> ...))`
  - Speciální objektová knihovna
  - Lze využít v předchozích pomocí `$<TARGET_OBJECTS:název>`

    ```
    add_executable(můj-cíl $<TARGET_OBJECTS:název>)
    ```
- Závislosti cílů
  - `target_link_libraries(<cíl> dep1 [dep2 ...])`
- Příklad

```
add_library(foo foo.cpp)
add_library(foo2 OBJECT foo2.cpp)
add_executable(bar main.cpp $<TARGET_OBJECTS:foo2>)
target_link_libraries(bar foo hdf5)
```

- Cíle pro testování
- Používané spolu s nástrojem CTest
- `add_test(NAME <název> COMMAND <příkaz> [args])`
  - Vytvoří cíl s názvem `<název>` pro testování
  - Pokud je `<příkaz>` názvem spustitelného cíle je jím nahrazen
  - Test lze spustit pomocí „`ctest -R <název>`“
- `set_tests_properties(<název> PROPERTIES <param>)`
  - Umožňuje nastavit parametry testu
  - `PASS_REGULAR_EXPRESSION` – výstup musí odpovídat R. V.
  - `TIMEOUT` – časový limit na provedení testu
  - `COST` – specifikuje pořadí testů
  - ...

```
add_executable(Scitani soucet.cpp)
add_test(Test Scitani 100 100)
add_tests_properties(Test PROPERTIES
    PASS_REGULAR_EXPRESSION "200")
```

CMake – vyhledávání knihoven

- Vyhledávání knihovny na základě jejího názvu
- `find_library(<VAR> název [cesta1 cesta2 ...])`
  - Hledá knihovnu „název“ ve výchozích a specifikovaných cestách
  - Výsledek uložen v proměnné CMake „`<VAR>`“
  - Pokud nebyla knihovna nalezena, nastaví se proměnná `<VAR>-NOTFOUND`
  - Cesty k nalezeným knihovnám uloženy v CMakeCache – úspěšné hledání se neopakuje!
- Příklady:

```
find_library(FOO_LIB foo)
find_library(FOO_LIB NAMES foo1 foo2)
find_library(FOO_LIB foo PATHS moje_cesta)
find_library(FOO_LIB foo ENV FOO_DIR NO_DEFAULT_PATH)
```

- Použití knihovny:
  - `target_link_libraries(můj_cíl ${FOO_LIB})`

CMake – vyhledávání balíčků

- Vyhledávání knihoven a nástrojů pomocí skriptů „FindXXX.cmake“
- Balíčky nastavují proměnné, přidávají cíle atd.
- `find_package(<BALÍČEK> ... [REQUIRED] ...)`
  - Hledá balíček pomocí modulu `FindBALÍČEK.cmake`
  - Modul musí být v některé z cest v `CMAKE_MODULE_PATH`
  - Proměnné a cíle, které modul nastavuje, jsou popsány v jeho dokumentaci (komentáři v *.cmake souboru)
  - Pokud není balíček nalezen, je nastavena proměnná `BALÍČEK_NOTFOUND`
- Příklady:

```
find_package(FOO)
find_package(FOO REQUIRED)
find_package(FOO NAMES foo1 foo2)
find_package(FOO 1.5 EXACT REQUIRED)
```

CMake – generování zdrojového kódu

- Umožňuje předat hodnoty proměnných CMake do zdrojových souborů
- `configure_file(<zdroj> <cíl> [ESCAPE_QUOTES])`
  - V souboru `<zdroj>` nahradí výskyty `@VAR@` a `${VAR}` za hodnotu proměnné `VAR` a výsledek uloží jako `<cíl>`
  - Umožňuje také nahradit `#cmakedefine VAR` za `#define VAR` nebo `/* #undef VAR */`
- Použití `configure_file` CMakeLists.txt

```
set(TEST_VAR "hodnota")
configure_file(soubor.cpp.in soubor.cpp)
```

soubor.cpp.in

```
#cmakedefine TEST_VAR
#define TEST_VAR "@TEST_VAR@"
#cmakedefine TEST_VAR_2
```

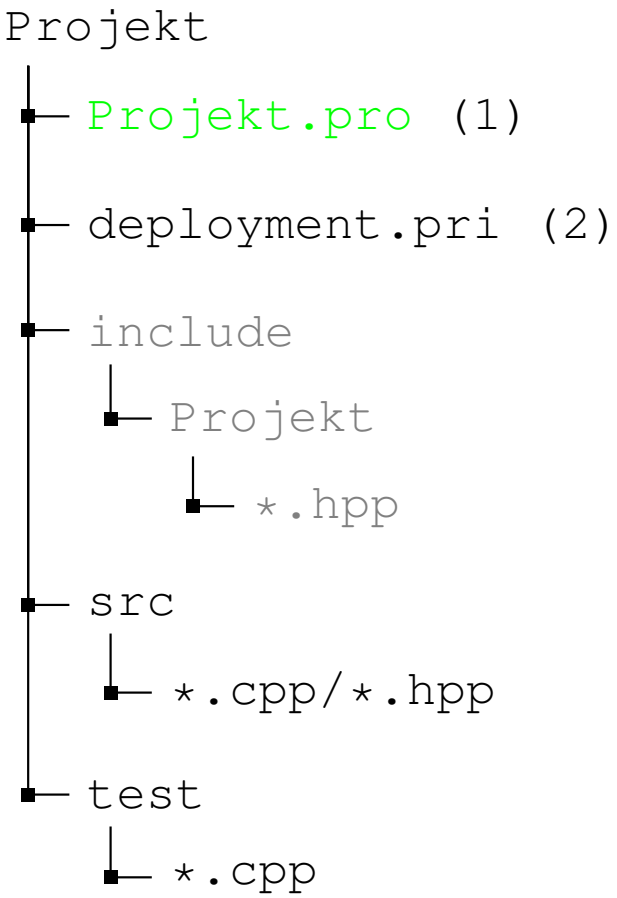
soubor.cpp

```
#define TEST_VAR
#define TEST_VAR "hodnota"
/* #undef TEST_VAR_2 */
```

- Projektově orientovaný meta sestavovací systém
- Multiplatformní, lze použít i pro projekt nevyužívající Qt
- Umožňuje generovat Makefile a projekty pro Visual Studio
- Zaměřený čistě na C/C++

QMake – Typická struktura projektu

- Popis projektu
  1. Centralizovaný popis projektu
  2. Případně doplněný o vkládané soubory
- Lze dosáhnout i adresářově orientovaného popisu (`SUBDIRS`)



QMake – základní syntaxe

- Základem syntaxe je přiřazení proměnné – specifikace cíle, konfigurace, šablony, atd.

```
<VAR>  = <hodnota1> [hodnota2 ...] # Přiřazení
<VAR> -= <hodnota1> [hodnota2 ...] # Odebrání
<VAR> += <hodnota1> [hodnota2 ...] # Připojení
```

- K proměnným lze přistupovat jako

```
$$<VAR> nebo $${<VAR>} # Proměnné QMake
$$(<ENV_VAR>)          # Proměnná prostředí
```

- Základní podmínky

```
win32 {
    release {
        # Windows - Release
    }
} else {
    # Jiný OS
}
```

```
win32:release {
    # Windows - Release
} else {
    # Jiný OS
}
```

QMake – typický projekt

- Soubor Projekt/Projekt.pro

```
TEMPLATE = app              # Typ projektu
CONFIG += console c++11 # Konzolová aplikace s C++11
CONFIG -= qt                # Bez Qt

TARGET = Projekt            # Hlavní cíl

INCLUDEPATH += . include/Projekt

# Hlavičkové a zdrojové soubory
HEADERS += include/Projekt/foo.h \
    include/Projekt/bar.h

SOURCES += src/main.cpp

include(deployment.pri)
```

- Soubor Projekt/deployment.pri

```
win32:release {
    target.path = $$OUT_PWD/deployment
    target.depends += install_helper

    export(target.path)

    install_helper.commands = \
        windeployqt $$OUT_PWD/release/$${TARGET}.exe
            -dir $$OUT_PWD/deployment

    INSTALLS += target
    QMAKE_EXTRA_TARGETS += install_helper
}
```

QMake – generátor sestavovacího systému

- `qmake [režim] [parametry] <soubory>`
  - Generování popisu sestavovacího systému (Makefile)

    ```
    > qmake -o <nazev-makefile> [cesta-k-pro]
    ```

  - Generování projektu/solution pro Visual Studio 2015
  - Parametr „`-tp`“ přidá prefix k `TEMPLATE` v projektu

    ```
    > qmake -spec win32-msvc2015 -tp vc [cesta-k-pro]
    ```

  - Parametrem „`-d`“ lze vynutit ladící výpisy (úrovně dle počtu jeho opakování)
  - Parametry „`-Wall, -Wparser, -Wlogic`“ lze povolit varování QMake

    ```
    > qmake -d -Wall [cesta-k-pro]
    ```

  - Lze nastavit hodnoty proměnných

    ```
    > qmake "VAR1=VALUE1" "VAR2=VALUE2" ... [cesta-k-pro]
    ```

- Prezentace – Filip Vaverka <ivaverka@fit.vutbr.cz>
- CMake
  - `https://cmake.org/cmake-tutorial/`
  - `https://cmake.org/cmake/help/latest/`
- QMake
  - `http://doc.qt.io/qt-5/qmake-manual.html`
- Ostatní
  - Ninja – `https://ninja-build.org/`
  - GYP – `https://gyp.gsrc.io/`
  - Blueprint – `https://github.com/google/blueprint`
  - Bazel – `https://bazel.build/`
  - Meson – `https://mesonbuild.com/`

imalik@fit.vutbr.cz