cmake

- CMake est un outil simplifié permettant la compilation de sources C et C++.
- C'est un outil multi-plateformes sous licence BSD
- Nécessite la présence d'un fichier **CMakeLists.txt**

- gestion automatique des dépendances
- Simple d'utilisation / facile à prendre en main

cmake : hello world !

- Un seul fichier main.c que l'on souhaite compiler en un programme « hello_world » :

```
project(HelloWorld)
cmake_minimum_required(VERSION 3.0)

add_executable(hello_world main.c)
```

- CMakeList.txt peut être découpé sur plusieurs répertoires avec des inclusions
- Un projet <<HELLO>> avec une bibliothèque dans le répertoire Hello et un programme d'exemple dans le répertoire Demo

cmake

- ./CMakeLists.txt

```
cmake_minimum_required (VERSION 3.0)
project (HELLO)

add_library(hello hello.c)

add_executable (helloDemo demo.c demo_b.c)
target_link_libraries (helloDemo hello)
```

- Si je structure mon projet en deux répertoires: *hello* pour la bibliothèque et *demo* pour un exemple d’utilisation...

cmake : utilisation

- cmake support l'out-source building : c'est à dire la compilation **en dehors** du répertoire des sources
- On suppose : projet/CMakeLists.txt
- alors on peut faire :

```
mkdir projet-build ; cd projet-build
cmake ../projet
make
```

- et

```
mkdir projet-debug ; cd projet-debug
cmake -DCMAKE_BUILD_TYPE=Debug ../projet
make
```

Exercices :

- Créer un fichier *CMakeLists.txt*
- Ajouter une règle pour créer l'exécutable « *example* » (celui qui n'utilise pas les bibliothèques)
- Tester votre compilation : créer un répertoire build et utiliser *cmake* depuis ce répertoire
- Ajouter une règle pour créer la bibliothèque **dynamique** (tester)
- Ajouter la règle pour créer l'exécutable utilisant cette bibliothèque et tester.
- Faire de même pour la bibliothèque **statique**

cmake

- ./CMakeLists.txt:

```
cmake_minimum_required (VERSION 3.0)
project (HELLO)

add_subdirectory (hello)
add_subdirectory (demo)
```

- ./hello/CMakeLists.txt

```
add_library (hello hello.c)
target_include_directories (hello PUBLIC ${CMAKE_CURRENT_SOURCE_DIR})
```

- ./demo/CMakeLists.txt

```
add_executable (helloDemo demo.c demo_b.c)
target_link_libraries (helloDemo LINK_PUBLIC hello)
```

cmake:cross platform make

- cmake est un système de compilation cross-plateformes. Il ne compile pas directement mais génère des fichiers dans différents formats :
  - Makefile
  - projet Visual Studio
  - Borland Makefile
  - projet Xcode
  - Kate
  - …
- cmake utilise les fichiers CMakeLists.txt et génère des fichiers en fonction de la plate-forme de compilation (Makefile, visual, xcode….).

cmake : les variables

- La déclaration de variables :
  - set(NAME VALUE)
  - ${NAME}
  - lors de l'appel à cmake : cmake -DNAME=VALUE
- Variables standards :
  - CMAKE_INCLUDE_PATH (pour les .h)
  - CMAKE_LIBRARY_PATH (pour la recherche de .so)
  - DESTDIR (pour l'installation)
  - CMAKE_BUILD_TYPE (Debug, Release)
- Dans le CMakeLists.txt :
  - CMAKE_C_FLAGS
  - CMAKE_C_FLAGS_DEBUG
  - CMAKE_C_FLAGS_RELEASE
  - CMAKE_CURRENT_SOURCE_DIR
  - CMAKE_CURRENT_BINARY_DIR
  - CMAKE_SOURCE_DIR

cmake : les fonctions

- add_executable(name sources)
- add_library(name STATIC sources)
- add_library(name SHARED sources)
- target_link_libraries(name libs)
- include_directories(dir1 dir2…)
- add_custom_command

Exercices :

- Organiser vos sources en répertoire : *hello* pour la bibliothèque et *src* pour le reste.
- Mettre à jour votre CMakeLists.txt