ISIMA – ZZ2 F2 et F3 - Architectures Logicielles et Qualité
Etude de cas – Réalisation d'un petit logiciel

Anaël Courjaud - 18/04/2023

Complément individuel : Description de l'outil CMake

1) *Courte présentation :*

CMake est un outil permettant de gérer de manière efficace et automatisée la compilation de projets logiciels. Ces projets peuvent être codés avec moult langages de programmation comme C, C++, Java, Python, etc ... et CMake est compatible avec différents environnements comme Linux, Windows, MacOS, iOS et Android. Le caractère open-source de l'outil ainsi que son utilisation de fichiers de configuration indépendants de la plateforme permet cette grande flexibilité et donc une meilleure portabilité.

Dans le cadre de notre projet, nous avons utilisé CMake pour faciliter la génération de builds en fonction des différentes versions du code.

2) *Procédure d'installation :*

CMake est disponible sur plusieurs plates-formes et peut être installé facilement. Personnellement, c'était déjà installé sur les machines de l'ISIMA, mais il est tout de même disponible en téléchargement sur le site officiel de CMake : www.cmake.org (gratuit). Vous pouvez également l'obtenir en tant que package sur les systèmes d'exploitation courants comme Linux, MacOS et Windows. Pour l'installation, il suffit de télécharger la version appropriée pour le système d'exploitation utilisé et d'ensuite suivre les instructions.

3) *Présentation des fonctionnalités :*

1. Configuration de projet : l'utilisateur doit d'abord configurer dans un fichier CMakeLists.txt les paramètres du projet, tels que les dépendances, les options de compilation, les bibliothèques externes, etc...
2. Cette configuration permet de générer automatiquement dans le répertoire courant des fichiers de configuration de build adaptés à la plateforme (un makefile pour un système Unix par exemple).

3. L’outil permet de gérer les dépendances, il offre une fonctionnalité permettant de télécharger et d'installer automatiquement les bibliothèques externes requises pour le projet.
4. CMake permet de personnaliser les builds pour différents environnements, plates-formes et outils de compilation. Il est par exemple possible de configurer une session de débogage dans l’IDE Visual Studio avec cet outil.
5. On peut intégrer CMake avec d'autres outils de développement tels que Git, SVN, Jenkins, etc...

4) *Mini-guide d'utilisation sur les principales fonctionnalités concernant le projet :*

Configuration :

Pour utiliser CMake, il faut tout d’abord configurer le projet. Pour cela, nous avons dû créer un fichier CMakeLists.txt dans le répertoire racine du projet puis y définir les informations suivantes :

1. Le nom du projet → définir avec la commande « project() ».
2. Les fichiers sources → définir avec « set() ».
3. Les bibliothèques externes → définir avec la commande « target_link_libraries() ».

Exemple de code pour le fichier CmakeLists.txt pour un projet comportant 3 fichiers de code (main.cpp, utils.cpp et son entête utils.hpp) et exploitant deux librairies (MyLibrary1 et MyLibrary2) :

```
cmake_minimum_required(VERSION 3.5)
project(MyProjectName VERSION 1.0.0)

# Définit les fichiers source de l'exécutable
set(SOURCES main.cpp utils.cpp utils.hpp)

set(LIBRARIES MyLibrary1 MyLibrary 2)

# Crée l'exécutable à partir des fichiers source
add_executable(MyProjectExecutable ${SOURCES})

target_link_libraries(MyProjectExecutable ${LIBRARIES})
```

Dans cet exemple, la commande `project()` est utilisée pour définir le nom du projet CMake ("MyProjectName") et sa version ("1.0.0"). Ensuite, la variable `SOURCES` est définie et l'ensemble des fichiers sources y sont affectés. Enfin, l'exécutable est créé avec la commande `add_executable()` qui prend en paramètre le nom que doit avoir l'exécutable ainsi que la variable `SOURCES`.

La commande `project()` sert également à définir les propriétés du projet, telles que les informations de version, le nom de l'entreprise/de l'organisation, la langue du projet, etc... Ces propriétés peuvent être utilisées dans d'autres parties du fichier CMakeLists.txt pour personnaliser la configuration et la génération de build.

Génération du build :

Après la configuration du projet via le fichier CMakeLists.txt vient la génération du build : dans le répertoire racine du projet, nous exécutons la commande « cmake -G "Unix Makefiles" » pour générer un build compatible Unix comportant un fichier Makefile.

Dans cette commande, on définit le générateur utilisé avec la mention "Unix Makefiles". Celui-ci est un des générateurs de build pris en charge par CMake. Il existe plusieurs autres générateurs de build disponibles, tels que :

1. Ninja : un générateur de build rapide et léger, particulièrement adapté aux projets de grande envergure.
2. Visual Studio : un générateur de build pour la plate-forme Windows, spécifiquement conçu pour Microsoft Visual Studio.
3. Xcode : un générateur de build pour la plate-forme Mac OS X, spécifiquement conçu pour Apple Xcode.
4. Eclipse CDT4 : un générateur de build pour l'environnement de développement Eclipse, spécialement conçu pour les projets C/C++.
5. Code::Blocks - Unix Makefiles : un générateur de build pour l'environnement de développement Code::Blocks, qui utilise le générateur de build "Unix Makefiles" de CMake.

Ces générateurs de build peuvent être spécifiés en utilisant l'option "-G" suivi du nom du générateur dans la commande de génération de build, par exemple "cmake -G Ninja". Une large gamme de générateurs de build est prise en charge par CMake pour répondre aux besoins de divers environnements, plates-formes et outils de compilation.

Compilation :

Enfin, pour compiler le projet dans le cadre d'un générateur Makefile, nous avons exécuté la commande « make », comme d'habitude.

Pour une configuration un peu plus avancée ... :

Gestion des dépendances :

Pour la gestion des dépendances, si une bibliothèque externe n'est pas disponible sur le système, CMake peut la télécharger et l'installer automatiquement. Pour cela, il faut ajouter certaines informations dans le fichier CmakeLists.txt.

Voici par exemple un exemple de code à inclure dans les fichier CMakeLists.txt pour l'ajout de la librairie MyLibrary1 disponible sur un dépôt GitHub donné :

```
include(FetchContent)
FetchContent_Declare(MyLibrary1 GIT_REPOSITORY https://github.com/*URL*)
```

Ajout d'options de compilation :

Pour ajouter des options de compilation telles que -Wall -Wextra, il faut utiliser la commande "add_compile_options()" dans le fichier CmakeLists.txt. Voici un exemple de code qui ajoute ces options à la compilation :

```
cmake_minimum_required(VERSION 3.5)
project(MyProjectName)

add_executable(MyProjectExecutable main.cpp)

# Ajoute les options de compilation -Wall et -Wextra
add_compile_options(-Wall -Wextra)

target_link_libraries(MyProjectExecutable MyLibrary1)
```

Les options de compilation ajoutées avec cette commande s'appliqueront à tous les fichiers source du projet. Pour rendre certaines options de compilation spécifiques à un fichier source ou à une bibliothèque particulière, il faut utiliser les commandes "target_compile_options()" et "add_library()".

Ce qu'on a utilisé au final :

Finalement, voici les lignes de code ajoutées à notre projet pour permettre l'utilisation de CMake.

Pour la configuration, voici notre fichier CmakeLists.txt :

```
cmake_minimum_required(VERSION 3.5)
project(ProjetFourmi VERSION 1.0.0)

set(SOURCES projetFourmi.cpp general.hpp fourmi.cpp fourmi.hpp caseMap.cpp
caseMap.hpp fourmilliere.cpp fourmilliere.hpp pheromone.cpp pheromone.hpp
manager.cpp manager.hpp mersenneTwister.cpp mersenneTwister.hpp)

add_executable(ExecutableProjetFourmi ${SOURCES})
add_compile_options(-Wall -Wextra)
```

Pour la génération, voici la commande : cmake -G "Unix Makefiles"

Pour la compilation, la commande make bien sûr !

5) *Synthèse des points clés du logiciel :*

Très simple d'installation, CMake est un outil open source multiplateforme de gestion de configuration et de génération de build qui permet de gérer de manière efficace et automatisée le processus de compilation de projets de logiciels. Les principales fonctionnalités de CMake incluent la configuration de projet, la génération de build, la gestion de la dépendance, la personnalisation des builds et l'intégration avec d'autres outils. Pour configurer un projet avec cet outil, il suffit de créer un fichier CMakeLists.txt dans le répertoire racine du projet et définir le nom du projet, les fichiers sources et les bibliothèques externes requises. Ensuite, le build est générable facilement en exécutant la commande "cmake". CMake est largement utilisé dans l'industrie pour la gestion de projets de logiciels et est considéré comme un outil essentiel pour les développeurs informatique.