Tutoriel CMake : Exercices

10 décembre 2012

Exercice 1 : Utiliser CMake

Le but de cet exercice est de comprendre l'utilisation de CMake en tant qu'outil de génération de Makefile, d'avoir un aperçu des autres fichiers/dossiers générés par CMake et de se familiariser avec la notion d'"outsource build".

En mode graphique

1. Aller dans le dossier exercice1 et prendre connaissance de son contenu :
   - un fichier `main.cpp`
   - un ficher `CMakeLists.txt`
2. Lancer CMake (en mode GUI)
3. Remplir le champ"Source directory"
4. Remplir le champ "Build directory" (créer un dossier "build", cf outsource build)
5. Cliquer sur "Configure" et choisir parmi les générateurs proposés : Linux/Mac => Unix Makefile, Windows => VisualStudio ou Mingw
6. Cliquer sur "Generate"
7. Compiler (make dans le dossier de build sous mac et linux, générer la solution sous visual studio)

En ligne de commande

Reprendre les étapes précédentes en ligne de commande. On utilisera ccmake pour générer le makefile.

Exercice 2 : Ecrire son premier CMakeLists.txt

Le but de cet exercice est d'écrire son premier CMakeLists.txt pour générer un makefile (ou une solution Visual Studio). On utilisera les commandes CMake : `add_executable`, `include_directories`, `add_library`, `option`, `link_directories` et `target_link_libraries`

1. Aller dans le dossier exercice2 et prendre connaissance de son contenu :
   - un fichier `main.cpp`
   - un fichier `foo.h`
   - un fichier `foo.cpp`
2. Ecrire un fichier CMakeLists.txt pour générer un binaire à partir de `main.cpp` et `foo.cpp`
3. Générer le makefile, compiler, exécuter
4. Reprendre les 2 points précédents en générant une bibliothèque "foo" à partir de `foo.cpp` et `foo.h` (On pourra tester avec `$BUILD_SHARED_LIBS` à ON et OFF).

Exercice 3 : Importer une bibliothèque externe

Le but de cet exercice est définir la bibliothèque "Foo" comme une bibliothèque externe qui sera importée pour compiler le projet "exercice3". On utilisera les commandes `find_package`, `find_path` et `find_library`.

1. Aller dans le dossier exercice3 et prendre connaissance de son contenu :

- un dossier `app` qui contient le projet "exercice3"
- un dossier `lib` qui contient le projet "FOO"

2. Ecrire le fichier CMakeLists.txt nécessaire pour génér le projet "FOO" qui correspond à la bibliothèque "foo" (on préfèrera une bibliothèque partagée)
3. Générer et compiler la bibliothèque "foo" dans un sous dossier 'build du dossier `foo`
4. Vérifier que tout est compilé (il doit y avoir un fichier libfoo.so quelque part dans le dossier de build, ou dans un sous dossier de build appelé "lib")
5. Remplir le fichier app/FindFOO.cmake
6. Ecrire le fichier CMakeLists.txt pour générer le projet "exercice3" (l'exécutable portera le même nom)
7. Dans un sous dossier `build` du dossier `app`, générer et compiler le makefile (attention, il faut renseigner des chemins pour que la configuration/compilation marche)

Exercice 4 : Lancer des tests et corriger les erreurs

Le but de cet exercice est de se familiariser avec l'outil `ctest` et avec l'application associée CDash.

1. Aller dans le dossier exercice4 et prendre connaissance de son contenu :
   - Un fichier `CMakeLists.txt` pour configurer l'ensemble du projet (ainsi que les tests)
   - Un fichier `CTestConfig.cmake` pour configurer les paramètres du serveur CDash
   - Les dossiers `app`, `bar`, `foo` et `tests`
2. Utiliser CMake pour configurer le projet. Activer la variable `BUILD_TESTING` en la mettant à ON.
3. Lancer `ctest` en mode "Continuous" et aller voir sur `http://cdash.inria.fr` (il y a un projet cmake-tutorial)
4. Corriger les erreurs !
5. Lancer `ctest` en mode "MemoryCheck"
6. Corriger l'erreur !

Exercice 5 : Générer un paquetage

Le but de cet exercice est de tester l'outil `cpack` pour la génération de paquetage (l'exemple pris sera celui d'un paquetage debian).

1. Aller dans le dossier exercice5 et prendre connaissance de son contenu
   - un fichier CMakeLists.txt
   - main.cpp
2. Configurer le projet avec CMake.
3. Compiler
4. Sous linux, générer le paquet Debian (Sous Windows utiliser le générateur NSIS (cf documentation) et sous MacOSX utiliser PackageMaker)