CMake

Atelier d'aide à la programmation

Léo Baudouin

baudouin.leo@gmail.com

03-04 juin 2019

Télécharger les exemples

Cloner le dépôt suivant

```
https://github.com/lbaudouin/module-cmake.git
```

Exemple 1 : Création d'un *Hello World* en C++

Fichier : **hello.cpp**

```
1 #include <iostream>
2 int main()
3 {
4    std::cout << "Hello world !" << std::endl;
5    return 0;
6 }
```

Exemple 1 : Création d'un *Hello World* en C++

Fichier : **hello.cpp**

```
1 #include <iostream>
2 int main()
3 {
4    std::cout << "Hello world !" << std::endl;
5    return 0;
6 }
```

Compiler et lancer le programme

```
g++ hello.cpp -o hello
./hello
```

Exemple 2 : Le Makefile

Permet de compiler automatiquement en faisant **make**

Template d'un Makefile

```
1 cible: dependances
2     commandes
```

Exemple 2 : Le Makefile

Permet de compiler automatiquement en faisant **make**

Template d'un Makefile

```
1 cible: dependances
2   commandes
```

Fichier : **Makefile**

```
1 hello: main.c hello.c
2   gcc -o hello hello.c main.c
```

Exemple 2 : Le Makefile

Permet de compiler automatiquement en faisant **make**

Template d'un Makefile

```
1 cible: dependances
2   commandes
```

Fichier : **Makefile**

```
1 hello: main.c hello.c
2   gcc -o hello hello.c main.c
```

Compiler et lancer le programme

```
make
./hello
```

Exemple 3 : Le Makefile

Fichier : **Makefile**

```
1  hello: hello.o main.o
2    gcc -o hello hello.o main.o
3
4  hello.o: hello.c
5    gcc -o hello.o -c hello.c -Wall
6
7  main.o: main.c hello.h
8    gcc -o main.o -c main.c -Wall
9
10 clean:
11   rm -rf *.o
```

Exemple 4 : Le Makefile avancé

Fichier : **Makefile**

```
1  CC = gcc
2  CFLAGS = -W -Wall
3
4  all: hello.o main.o
5    $(CC) -o hello $^
6
7  %.o: %.c
8    $(CC) -c $< -o $@ $(CFLAGS)
9
10 clean:
11   rm -rf *.o
```

Exemple 5 : CMake

Fichier : **CMakeLists.txt**

```
1 ADD_EXECUTABLE(hello main.c hello.c)
```

Compiler et lancer le programme

```
mkdir build; cd build
cmake ..
make
./hello
```

A voir

Ouvrir le fichier Makefile généré

Astuce

CMake

- Utilisation d'un dossier de **build**
- Utilisation des variables :
  `cmake -DCMAKE_BUILD_TYPE=Release ..`
- Gestion des variables/options avec **ccmake** ou **cmake-gui**

Make

- Utiliser plusieurs threads : `make -j 5`
- Forcer la recompilation : `make -B`
- Passer les erreurs : `make -k`

Utilisation de CMake

Cloner le dépôt suivant

```
https://gitlab.com/lbaudouin/libempty.git
```

Arborescence

```
|-- bin
|   |-- CMakeLists.txt        < Règles de compilations des binaires
|   |-- main.cpp              < Source d'un binaire
|-- CMakeLists.txt            < Règles générales
|-- doc
|   |-- doc.tag               < Fichier de documentation
|   |-- Doxyfile.in           < Règle de la documentation
|   |-- style.css             < Style pour la documentation
|-- include
|   |-- my_class.h            < Fichiers d'en-tête
|-- LibraryConfig.cmake.in    < Fichier de configuration pour cmake
|-- Library.pc.in             < Fichier de configuration pour pkg-config
|-- LICENSE                   < Licence du projet
|-- README                    < Aide pour l'utilisateur
|-- src
|   |-- my_class.cpp          < Sources du projet
|-- unitTesting               < Tests unitaires
    |-- CMakeLists.txt        < Règles de compilations des tests unitaires
    |-- TestPass.cpp          < Test unitaire pour une seule fonctionnalité
```

Compilation

Réaliser les étapes suivantes :

1. `mkdir build; cd build`
2. `cmake ..`
3. `make && make install`

Compilation

Réaliser les étapes suivantes :

1. `mkdir build; cd build`
2. `cmake ..`
3. `make && make install`

4. `ccmake ..`
5. Modifier les options suivantes :
   (BUILD_TEST, CMAKE_BUILD_TYPE, CMAKE_INSTALL_PREFIX)
6. `make install`
7. `make test`

Compilation

Réaliser les étapes suivantes :

1. `mkdir build; cd build`
2. `cmake ..`
3. `make && make install`

5. `ccmake ..`
6. Modifier les options suivantes :
   (BUILD_TEST, CMAKE_BUILD_TYPE, CMAKE_INSTALL_PREFIX)
7. `make install`
8. `make test`

9. `make doc`

Création du CMakeLists.txt

Objectifs

- Compiler un binaire
- Compiler un test unitaire

Création du CMakeLists.txt

Objectifs

- Compiler un binaire
- Compiler un test unitaire

- Compiler une bibliothèque (library)
- Installer la bibliothèque et le binaire
- Ajouter des options de compilation
- Générer la documentation

Fonctions

Généralités

- Mettre des commentaires :
  `#Commentaire`
- Version minimal de CMake :
  `CMAKE_MINIMUM_REQUIRED(VERSION 2.6)`
- Définir un projet :
  `PROJECT(project_name)`
- Définir une variable :
  `SET(variable_name "value")`
- Récupérer la valeur d'une variable :
  `${variable_name}`
- Créer des options :
  `OPTION(variable_name "Description" default_value)`

Consignes

Créer un projet

- Ouvrir le dossier `example`
- Créer un fichier `CMakeLists.txt`
- Définir la version minimal de cmake (ex : 2.8)
- Définir un nouveau projet « hello »

Fonctions

Créer des cibles

- Ajouter un exécutable :
  `ADD_EXECUTABLE( exec_name sources )`

Tests

- Activer les tests :
  `ENABLE_TESTING()`
- Ajouter un test :
  `ADD_TEST( test_name exec_name)`

Consignes

Créer les cibles

- Générer un exécutable `hello`
- Générer un test `test_hello`
- Compiler
- Lancer le programme
- Lancer le test

Fonctions

Créer des cibles

- Ajouter une bibliothèque :
  `ADD_LIBRARY( lib_name [static|shared] sources )`
- Linker une cible avec une bibliothèque :
  `TARGET_LINK_LIBRARIES( exec_name libraries )`

Sous-dossier

- Inclure un sous-dossier :
  `ADD_SUBDIRECTORY( folder )`

Dossier d'en-tête

- Ajouter un dossier d'en-tête :
  `INCLUDE_DIRECTORIES( folder )`

Consignes

Créer les cibles

- Créer les dossier `bin`, `lib` et `test`
- Répartir les fichiers
- Créer la bibliothèque `hello`
- Créer le binaire `hello`
- Créer le test unitaire

Dépendances

Utiliser une bibliothèque

- Chercher une bibliothèque :
  `FIND_PACKAGE(package_name [version] [EXACT] [REQUIRED])`
  Cherche les fichiers :
  - `package_nameConfig.cmake`
  - `package_name-config.cmake`
- Savoir si la bibliothèque a été trouvée :
  `package_name_FOUND`
- Ajouter des dossier d'includes :
  `include_directories( <directory> )`

Dépendances

Utiliser une bibliothèque

- `XXX_FOUND` :
  Passe à TRUE si la bibliothèque XXX a été trouvée
- `XXX_YYY_FOUND` :
  Passe à TRUE si le composant YYY de XXX a été trouvé
- `XXX_INCLUDE_DIRS` :
  Liste les dossiers d'include
- `XXX_LIBRARIES` :
  Liste les fichiers contenant les bibliothèques (chemin complet)
- `XXX_LIBRARY_DIRS` : (Optionnel)
  Liste les dossiers contenant les bibliothèques
- `XXX_DEFINITIONS` :
  Liste les définitions pour la bibliothèque

Dépendances

Utiliser une bibliothèque avec un fichier .pc

```
SET(XXX_FOUND FALSE)
FIND_PACKAGE(PkgConfig REQUIRED)
IF(PKG_CONFIG_FOUND)
  PKG_CHECK_MODULES(XXX REQUIRED xxx>=1.8.0)
ENDIF()

IF(XXX_FOUND)
  MESSAGE("XXX found")
ELSE()
  MESSAGE("XXX>=1.8.0 not found")
ENDIF()
```

Consignes

Générer la doc

- Chercher le package **Doxygen** avec `FIND_PACKAGE`
- Tester si Doxygen a été trouvé
- Configurer le fichier `${CMAKE_SOURCE_DIR}/doc/Doxyfile.in` et le sortir dans `${PROJECT_BINARY_DIR}/doc/Doxyfile`
- Créer une cible personnalisée "doc" exécutant `${DOXYGEN_EXECUTABLE}`