Tutoriel: Compiler un programme en C++

Ce tutoriel propose de reprendre dans deux cas de figure (un unique fichier source, et plusieurs fichiers sources en C++11) différentes approches permettant de compiler des fichiers en C++.

Contents

1- Compilation d'un fichier unique

Soit un programme d'exemple contenant un unique fichier (`main.cpp` par exemple).

```
#include <iostream>
#include <string>

int main()
{
  std::string variable="Mon premier";
  variable += " programme.";

  std::cout<<variable<<std::endl;

  return 0;
}
```

Il est possible de compiler ce fichier suivant différentes approches.

1.1 En ligne de commande

Cette solution est possible lorsque vous avez un seul fichier source, elle est à déconseiller si votre programme contient plusieurs fichiers, car trop répétitive.

Ouvrez une ligne de commande dans le même répertoire que le fichier `main.cpp`. Tapez

```
g++ main.cpp -g -Wall -Wextra -o main
```

Cette commande vient générer le fichier `main` exécutable qui se lance dans le même terminal (observez le à l'aide de la commande `ls`) à l'aide de la commande

```
./main
```

Notez que dans le cas d'un fichier nécessitant C++11, on écrira alors

```
g++ main.cpp -g -Wall -Wextra -std=c++11 -o main
```

1.2 À l'aide d'un Makefile

Solution généralement conseillée: précision et compréhension de la démarche.

Dans le même répertoire que le fichier `main.cpp`, créer un fichier du nom de `Makefile`, et remplissez le de la manière suivante à l'aide d'un éditeur de texte.

```
CXXFLAGS = -g -Wall -Wextra -std=c++11

all: main
main : main.cpp
clean:
    rm -f *~ main
```

Notez qu'il y a une tabulation avant la commande `rm`, et non une suite d'espaces.

Pour compiler le programme, tapez ensuite dans la ligne de commande

```
make
```

Cette commande vient générer l'exécutable `main` que vous pouvez ensuite lancer.
Notez qu'il est possible de *nettoyer* le répertoire des fichiers temporaires et de l'exécutable par l'appel à la commande

```
make clean
```

Il est par contre inutile d'appeler `make clean` entre deux compilations.

1.3 À l'aide de QMake

Solution envisageable pour un projet utilisant Qt. Il n'est pas conseillé d'utiliser QMake sur un autre type de projet: peu de contrôles et ajout de bibliothèques et dépendances inutiles.

QMake est un outil associé à la bibliothèque de développement Qt permettant de générer sans effort des Makefile à partir d'un ensemble de fichiers sources.

Dans un répertoire en amont de celui contenant le fichier `main.cpp`[1], tapez en ligne de commande

```
qmake -project
```

Observez la création d'un fichier du type `[nom_repertoire].pro`. Ce fichier est un fichier de configuration pour l'outil `QMake`. Cette première étape permet de parcourir les répertoires du projet pour générer ce fichier de configuration. Il est cependant également possible d'écrire ce fichier entièrement à la main pour d'autres projets.

Dans le cas présent, les Warnings ne sont pas activés. Pour activer ceux-ci, il est possible d'éditer le fichier `[nom_repertoire].pro` à l'aide d'un éditeur de texte, et d'y ajouter la ligne suivante:

```
QMAKE_CXXFLAGS += -g -Wall -Wextra -std=c++11
```

Tapez ensuite en ligne de commande

[1] Il est possible de faire cette démarche dans le même répertoire, mais il est conseillé d'éviter cette pratique pour éviter de *polluer* votre répertoire source et d'écraser un Makefile existant

```
qmake .
```

Cette commande permet de générer un Makefile à partir du fichier de configuration `.pro`. De manière traditionnelle, la compilation puis l'exécution se réalisent à l'aide des commandes suivantes:

```
make
./[nom_repertoire]
```

1.4 À l'aide de CMake

Solution envisageable pour un projet quelconque utilisant Qt ou d'autres bibliothèques. Il est recommandé de connaitre l'utilisation et la configuration d'un projet CMake. CMake génère cependant un Makefile peu lisible, moins générique et inclus de nombreuses dépendances parfois inutiles. Il reste ainsi conseillé de pouvoir également être capable de générer un Makefile plus simple équivalent.

CMake est un outil permettant de simplifier la création d'un Makefile pour un projet existant. CMake est développé par Kitware [2] qui l'utilise classiquement pour compiler leurs projets (VTK, ITK, etc).

Dans un répertoire en amont de celui contenant le fichier `main.cpp`[3], créez un fichier `CMakeLists.txt` qui contient les informations suivantes

```
cmake_minimum_required(VERSION 2.6)

project(main)

set(CMAKE_BUILD_TYPE debug)
set(CMAKE_CXX_FLAGS "-Wall -Wextra -std=c++11")

file(
    GLOB_RECURSE
    source_files
    src/*.cpp
)

add_executable(
    main
    ${source_files}
)
```

Ce fichier est un fichier de configuration que l'on écrit à la main. L'avantage de ce fichier concerne la description récursive et générique des noms de fichiers. CMake possède également l'avantage de pouvoir trouver certaines librairies de manière automatique sur différents PC.

Tapez en ligne de commande

[2] société de développement de logiciels en imagerie médicale et informatique scientifique
[3] Idem que pour QMake

```
cmake .
```

Cette commande a pour effet de lancer l'outil `CMake` sur le fichier de configuration `CMakeLists.txt` et génère un fichier `Makefile`.

Note: Il est également possible d'avoir un éditeur configurable en mode graphique en tapant `cmake-gui ..`

Une fois le `Makefile` généré, celui-ci permet de compiler puis d'exécuter le projet de manière traditionnelle:

```
make
./main
```

Note: L'utilisation de `CMake` génère plusieurs fichiers temporaires: `CMakeCache.txt`, `cmake_install.cmake`, ainsi que le `Makefile`, et un répertoire temporaire `CMakeFiles/`. Ces fichiers et ce répertoire peuvent être supprimés sans craintes, le seul fichier à garder est le fichier de configuration `CMakeLists.txt`. En particulier, il est recommandé de supprimer tous les fichiers temporaires lorsque vous déplacer votre projet, ou bien changez de machine. Il suffit alors de relancer l'outil `CMake` sur votre fichier de configuration pour créer un nouveau `Makefile`.

2- Compilation de plusieurs fichiers

Supposons désormais les fichiers suivants dans le même répertoire:

```cpp
//FICHIER main.cpp
#include "lib1.hpp"
#include <iostream>
#include <string>

int main()
{
  std::cout<<"J'appelle ma fonction:"<<std::endl;
  ma_fonction();

  return 0;
}
```

```cpp
//FICHIER lib1.hpp
#pragma once

#ifndef LIB1_HPP
#define LIB1_HPP

void ma_fonction();

#endif
```

```cpp
//FICHIER lib1.cpp
#include "lib1.hpp"
#include <iostream>

using std::cout;
using std::endl;

void ma_fonction()
{
    auto T={7,8,9,-12};
    for(auto value : T)
        cout<<value<<endl;
}
```

Nous pouvons également détailler les différentes approches de compilation.

2.1 En ligne de commande

Ouvrez une ligne de commande dans le même répertoire que le fichier `main.cpp`. Tapez dans un ordre quelconque la demande de compilation des deux fichiers

```
g++ -c lib1.cpp -g -Wall -Wextra -std=c++11
g++ -c main.cpp -g -Wall -Wextra -std=c++11
```

Réalisez ensuite l'édition de lien pour créer l'exécutable

```
g++ lib1.o main.o -o main
```

L'executable généré peut alors être lancé par la commande

```
./main
```

2.2 À l'aide d'un Makefile

Dans le même répertoire que vos fichiers sources, écrivez cette fois le Makefile suivant:

```
CXXFLAGS = -g -Wall -Wextra -std=c++11
CXX=g++

#executable name
project=main

all: ${project}

#linking
${project} : main.o lib1.o
    ${CXX} $^ -o ${project}

#compile object files
main.o: main.cpp lib1.hpp
lib1.o: lib1.cpp lib1.hpp

clean:
    rm -f *~ *.o ${project}
```

Puis demandez la compilation et enfin l'exécution du projet par les deux lignes suivantes:

```
make
./main
```

2.3 À l'aide de QMake

Créez tout d'abord le fichier de configuration `.pro` dans un répertoire en amont de vos sources.

```
qmake -project
```

Éditez ensuite le fichier de configuration `.pro` afin d'y ajouter les paramètres suivants

```
QMAKE_CXXFLAGS += -g -Wall -Wextra -std=c++11
```

Puis créez le Makefile, compilez et exécutez le projet à l'aide des commandes suivante

```
qmake .
make
./[nom_repertoire]
```

2.4 À l'aide de CMake

Dans un répertoire en amont de vos fichiers sources, créez cette fois un fichier `CMakeLists.txt` avec les instructions suivantes.

```
cmake_minimum_required(VERSION 2.6)

project(main)

set(CMAKE_BUILD_TYPE debug)
set(CMAKE_CXX_FLAGS "-Wall -Wextra -std=c++11")

file(
    GLOB_RECURSE
    source_files
    src/*.[ch]pp
)

add_executable(
    main
    ${source_files}
)
```

Remarque: ce fichier est très générique et peux être recopié tel pour compiler de nombreux projets.

Faites ensuite appel à `CMake` pour générer le `Makefile`, compilez le projet, puis exécuter celui-ci à l'aide des 3 commandes suivantes.

```
cmake .
make
./main
```