

Compilation en ligne de commande avec G++

François FOUQUET

1^{er} février 2012

Liris - CNRS Lyon

- 1 Introduction : le compilateur
 - Rôle du compilateur
 - La compilation
- 2 Compilation d'un fichier
 - Commandes de base
- 3 Compilation d'un ensemble de fichiers
 - Compilation globale
 - Compilation séparée
- 4 options de compilation

Compilateur ?

Le compilateur ...

- C'est un programme (très complexe) qui sert à traduire le code écrit en langage de programmation (en C++ dans notre cas) en code exécutable.
- Il génère un fichier exécutable contenant le code machine (binaire) de votre programme.
- Il génèrera des exécutables **différents** en fonction de la configuration (matérielle et logicielle) de la machine cible.

Attention !

Le compilateur n'est pas un outil "magique". Ce n'est pas parce que votre code compile sans erreur qu'il ne fera pas n'importe quoi en cours d'exécution ! Il se contente de vérifier la conformité à la norme de ce que vous avez écrit et de le traduire.

Compilateur vs Interpréteur

- Il existe des langages qui n'ont pas besoin d'être compilé (ex : python).
- Pour ces langages, un interpréteur se charge de traduire le source en code machine à mesure qu'il doit être exécuté.
 - plus grande souplesse pour le programmeur (pas besoin de recompiler à chaque modification).
 - l'application est "re-traduite" à chaque fois qu'elle est lancée (peut dans certains cas engendrer des baisse de performances).

Étapes de la compilation avec G++

Le compilateur G++ procède en 4 étapes pour compiler un fichier source écrit en C++ :

- La précompilation
- La compilation
- L'assemblage
- L'édition de liens

Pour en savoir plus :

<http://www.cmi.univ-mrs.fr/~contensi/coursC/index.php?section=env&page=comp>

Précompilation

- Réalisé par un pré-compilateur.
- Traite toutes les directive commençant par `#` dans votre code et transforme le fichier à compiler en conséquence.
- Ex : `#include < iostream >` va copier tout le contenu du fichier `iostream` dans votre fichier source afin que les fonctions qui y sont définies soient utilisable dans votre programme.

Compilation

- **Traduction du code source en code assembleur.**
- Le compilateur vérifie la conformité du code aux règles de programmation en C++
- Il affiche à l'utilisateur les éventuelles erreurs et des warnings (si l'option est activée).
- Les erreurs sont bloquantes (aucun fichier exécutable n'est généré), mais pas les warnings.

Assemblage

- **Traduction du code assembleur en code binaire exécutable par la machine cible.**
- Sauf cas exceptionnels, cette étape ne pose jamais de problème...

Édition de liens

- Nécessaire si le programme à construire provient de plusieurs fichiers sources.
- Réalisé par l'éditeur de lien ou linker.
- Regroupe les binaires créés pour chaque fichier source en un seul exécutable.
- Crée les liens entre les endroits des binaires où les fonctions sont définies et les endroits où elles sont appelées
- Peut rechercher le code exécutable de certaines fonctions dans des bibliothèques de fonctions (library) passées en paramètre.
- Affiche des erreurs à l'utilisateur si le code exécutable d'une fonction appelée est introuvable dans les binaires scannés.

- 1 Introduction : le compilateur
 - Rôle du compilateur
 - La compilation
- 2 Compilation d'un fichier
 - Commandes de base
- 3 Compilation d'un ensemble de fichiers
 - Compilation globale
 - Compilation séparée
- 4 options de compilation

Compilation simple

Pour réaliser les 4 étapes de compilation sur un fichier source MonSource.cpp on utilise g++ :

Compilation d'un fichier

```
g++ MonSource.cpp
```

Par défaut, l'exécutable généré se nommera a.out.

Attention !

Pour que la compilation fonctionne votre fichier doit obligatoirement contenir une et une seule fonction **main** !

Compilation simple

Pour changer le nom du fichier de sortie, il faut utiliser l'option `-o` suivie de nom de l'exécutable voulu.

Compilation d'un fichier avec choix du nom de l'exécutable

```
g++ MonSource.cpp -o NomExecutable
```

- 1 Introduction : le compilateur
 - Rôle du compilateur
 - La compilation
- 2 Compilation d'un fichier
 - Commandes de base
- 3 Compilation d'un ensemble de fichiers
 - Compilation globale
 - Compilation séparée
- 4 options de compilation

Compilation globale

Pour compiler un programme composé de plusieurs fichiers sources, il suffit de passer tout ces fichiers en une fois au compilateur :

Compilation globale

```
g++ MonSource1.cpp MonSource2.cpp MonSource3.cpp -o NomExecutable
```

- On ne passe que les fichier .cpp (Normalement les .h sont inclus dans les .cpp)
- Les fichiers sont **tous** compilé à chaque fois (même ceux qui n'ont pas changés).
- La compilation globale prend beaucoup de temps.
- On lui préfère la compilation séparée.

Principes

- Pour éviter de recompiler tous les fichiers à chaque fois qu'on en modifie uniquement un.
- **Principe : Séparer la création des fichiers binaires associés à chaque fichier source de l'édition de lien.**
- On crée explicitement et on stocke un fichier binaire (.o) pour chaque fichier source (.cpp).
- Seuls les binaires des fichiers sources qui ont été modifiés sont à régénérés.
- L'édition de lien est ensuite refaite avec tous les fichiers.
- inconvénient : on doit appeler autant de fois la commande `g++` qu'on a de fichiers à compiler (pour l'éviter : faire des Makefiles ...)

Mise en pratique : génération des binaires

En pratique, on générera les binaires des fichiers sources en passant l'option -c à g++ :

Compilation séparée : Génération des binaires

```
g++ -c MonSource1.cpp MonSource3.cpp
```

Par défaut g++ nomme les fichiers .o avec le même nom que les fichiers .cpp correspondant (seule l'extension change).
Il est toujours possible de changer le nom du fichier de sortie avec l'option -o (ne fonctionne qu'avec un seul fichier en entrée)

Mise en pratique : Édition des liens

Pour faire l'édition de lien on passe simplement à g++ l'ensemble des binaires à associer dans l'exécutable.

Compilation séparée : Édition des liens

```
g++ MonBinaire1.o MonBinaire2.o MonBinaire3.o -o NomExecutable
```

Attention !

Que vous utilisiez la compilation globale ou la compilation séparée, seul un (et un seul) de vos fichiers sources doit contenir une fonction nommée **main**. Il est conseillé de mettre cette fonction seule dans un fichier qu'on nommera main.cpp.

- 1 Introduction : le compilateur
 - Rôle du compilateur
 - La compilation
- 2 Compilation d'un fichier
 - Commandes de base
- 3 Compilation d'un ensemble de fichiers
 - Compilation globale
 - Compilation séparée
- 4 options de compilation

Affichage des warnings

Sans option le compilateur est "silencieux" sauf en cas d'erreur.

Il est souvent utile d'afficher les warnings :

- -Wall : affiche les warnings courants.
- -Wextra : affiche des warnings supplémentaires.
- -pedantic : vérifie entièrement la conformité du code à la norme.

Si votre code compile sans warning avec l'option -pedantic, il a de grandes chances d'être portable.

Autres options utiles

g++ propose plusieurs milliers d'options dont notamment :

- -g : Ajoute des symboles de debug dans votre exécutable (nécessaires pour exécuter le programme dans un débbugger).
- -pg : Ajoute les instructions de profiling (pour analyser les performances avec gprof)
- -o2 : Active les optimisations de compilation pour rendre votre programme plus performant.
- -I (i maj.) : Ajoute un répertoire où le pré-compilateur recherchera les fichiers d'include (.h / .hpp).
- -l : Ajoute une bibliothèque de fonctions où le linker cherchera les binaires des fonctions "externes" appelées par le programme.
- -L : Ajoute un répertoire où le linker recherchera les bibliothèques de fonctions passées avec -l.