# Présentation g++

Mathieu ARQUILLIERE

April 8, 2020

# Contents

# 1 Présentation

La *compilation* est l'ensemble des étapes qui permettent de convertir le code source d'un programme, en un fichier compréhensible par une machine. Il y a plusieurs étapes lors de la compilation. La première est l'étape de préprocesseur, on vient remplacer les directives (les lignes commençant par # en C et C++). Ensuite le compilateur vérifie que le code source est valide, si c'est le cas il procédera alors à l'étape de compilation des sources en fichiers objets. Enfin il restera à rassembler les fichiers pour ne former qu'un exécutable (assemblage).

Nous allons ici étudier le compilateur linux g++ qui se charge de compiler des sources provenant du C++. Ce compilateur est disponnible sous Linux, et est le compilateur par défaut dans beaucoup de situation.

# 2 Utilisation

La commande basique pour compiler avec cet outils est la suivante :

```
g++ source.cpp
```

En supposant que la compilation ne lévera pas d'erreur, cette commande créera un fichier de sortie appelé a.out (l'exécutable). Une option permet de changer le nom du fichier de sortie.

```
g++ -o executable source.cpp
```

On peut alors ajouter plusieurs fichiers sources en les rajoutants les uns à la suite des autres.

```
g++ -o executable source_1.cpp source_2.cpp
```

# 3 Paramètres sur les avertissements

Pour un développeur, les avertissements sont très important car ils font très souvent ressortir des erreurs de programmations que l'on n'a pas vu, et que le compilateur peut tout à fait compiler. Il peut donc être intéressant de chercher à en activer plus que d'origine.

## 3.1 Les options populaires

On retrouvera ces options dans les chaînes de compilation de la majorité des projets. En effet elles permettent de montrer au développeur presque toutes les erreurs qui peuvent s'être glissées dans son code. Ces options sont en réalité des raccourcis pour éviter d'ajouter plusieurs options. On peut trouver la liste complète des options activées par ces deux raccourcis dans le manuel.

```
g++ -Wall -Wextra source.cpp
```

La première est contre intuitive puisqu'elle n'ajoute pas tous les avertissements possibles, mais seulement les plus courant. Par exemple grâce à elle un avertissement sera affiché quand l'initialisation des attributs ne se fait pas dans le même ordre que leur déclaration. Les variables et fonctions non utilisées causeront également des avertissements. La seconde cependant correspond bien à son nom (enseignement appelé -W). On aura ici un avertissement si un bloc if/else ne contient aucune instruction.

### 3.2 Version du langage

Il est possible d'afficher des avertissements lorsque une fonctionnalité que vous utilisiez a été changé pour un standard du langage différent. Cela perme de s'assurer préventivement qu'un changement de standard n'entrainera pas un problème majeur de compilation. Pour le C++, cette option existe pour 3 standards : 11, 14 et 17.

```
g++ -Wc++17-compat source.cpp
```

### 3.3 Autres options utiles

Cette option permet de prévenir qu'une fonction qui a été déclarée *inline* n'a pas pu être compilée comme tel.

```
g++ -Winline source.cpp
```

L'option suivante permet de s'assurer qu'aucune conversion dangereuse n'est faite sur des pointeurs.

```
g++ -Wcast-qual source.cpp
```

Enfin cette dernière option permet de vérifier que les conditions de blocs if/else-if ne soient pas identiques.

```
g++ -Wduplicated-cond source.cpp
```

## 4 Paramètres pour le debug

Le compilateur permet de compiler le programme dans le but de le tester avec un debugger. Pour celà il faut changer l'exécutable pour qu'il contienne des informations sur le code source. On peut utiliser plusieurs options pour ajouter ces informations.

```
g++ -g source.cpp -> Informations de debug dans le format natif
g++ -ggdb source.cpp -> Informations de debug pour gdb
g++ -gdwarf source.cpp -> Informations de debug pour dwarf
g++ -gstabs source.cpp -> Informations de debug pour stabs
```

Il est également possible de spécifié la quantité d'informations ajoutées grâce à l'option suivante. On peut mettre un niveau allant de 0 (aucune information) à 3 (toutes les informations). Evidemment cette option fonctionne avec n'importe lequel des debugger choisis, et la description de chaque niveau est précisée dans le manuel.

```
g++ -g -glevel3 source.cpp
```

## 5 Paramètres pour l'optimisation

Il est possible d'optimiser le programme de sortie selon plusieurs axes. Les premières options permettent de réduire le temps d'exécution du programme. On privilégiera le second niveau d'optimisation, le troisième pouvant causé des problèmes (surtout lorsque des calculs précis doivent être fais). Il existe donc les niveaux 1, 2 et 3.

```
g++ -O2 source.cpp
```

Un niveau spécial peut être utilisé lors du debug, il est d'ailleurs recommandé pour ce cas de figure. Il ajoutera toutes les optimisations de premier niveau qui n'interfèrent pas avec le debbug.

```
g++ -Og -glevel3 source.cpp
```

Enfin une dernière option existe pour réduire la taille de l'exécutable plutôt que de privilégier les performances. Cette option est surtout utilisée lors du développement dans le monde de l'embarqué ou de l'internet des objets.

```
g++ -Os -glevel3 source.cpp
```

L'ensemble des options présentées précédemment sont à la manière de -Wall et -Wextra, des raccourcis regroupand une grande quantité d'options. Il est donc possible de choisir exactement les options que l'on veut ajouter pour l'optimisation. Le manuel est encore une fois très utile pour avoir des renseignements plus complets.