

## Compilation séparée et options de gcc

1. Chaîne de compilation multi-modules .....	1
2. Principales options du compilateur gcc.....	2
3. Exemple .....	3

### 1. Chaîne de compilation multi-modules

Compiler un programme C nécessite un processus en *deux temps*<sup>1</sup>.

Dans une première étape dite de « compilation » proprement dit, on va compiler les uns après les autres les fichiers source .c de chacun des modules.

Cette étape de compilation produit, *pour chaque fichier .c*, un fichier .o de même nom que le fichier .c qui contient une traduction du code C en langage machine binaire.

Un fichier objet .o n'est pas exécutable : il ne contient que la traduction du code du module, dont les appels des fonctions d'autres modules, mais en aucun cas le binaire de ces fonctions.

Dans une seconde étape, dite « d'édition des liens » (*linking*), le compilateur va assembler l'ensemble des fichiers objets .o issus de l'étape de compilation ainsi que les binaires des bibliothèques qu'utilise le programme. Le résultat est un fichier binaire exécutable, qui contient toutes les informations nécessaires au lancement du programme.

C'est lors de cette seconde étape que le compilateur vérifie qu'il dispose bien des codes binaires de toutes les fonctions utilisées par le programme.

L'étape de *compilation* est lancée avec l'option `-c` du compilateur gcc. Par exemple :

```
$ gcc -c unModule.c
```

produira (si il n'y a pas d'erreur de compilation) un fichier binaire `unModule.o`

L'étape *d'édition des liens* est lancée avec l'option `-o` du compilateur gcc.

Par exemple :

```
$ gcc -o monProgramme unFichier.o unAutre.o unTroisieme.o -lm
```

assemble les binaires des modules `unFichier.o`, `unAutre.o` et `unTroisieme.o` ainsi que le binaire de la bibliothèque mathématique (« m ») et produira (si il n'y a pas d'erreur) un fichier binaire exécutable nommé `monProgramme`.

---

<sup>1</sup> Certains d'entre vous ne le savent peut être pas encore : dans des cas simples tels que ceux que vous avez vu en première année, les deux étapes sont parfois regroupées. C'est fini, désormais !

## 2. Principales options du compilateur gcc

On présente succinctement dans les tableaux qui suivent quelques unes des très nombreuses options du compilateur gcc – celles auxquelles vous pourriez avoir affaire. Voir, bien sûr, man gcc pour plus de précision.

### Choix de l'étape (compilation / édition des liens)

-c	Etape de compilation. Exemple : <code>gcc -c unModule.c</code> produit le fichier unModule.o
-o <output>	Etape d'édition des liens. Exemple : <code>gcc -o monProgramme unFichier.o autre.o troisieme.o -lm</code> produit le fichier exécutable monProgramme en liant les différents binaires.

### Options propre à l'étape de compilation (-c)

-Wall	« Warn All » : affiche un warning pour tout problème potentiel détecté. Bien utile...
-g	Compilation en mode « debug » : ajoutera au binaire de nombreuses informations utiles au debugger. Ces ajouts vont ralentir le programme, mais permettre de le lancer dans un debugger tel que gdb ou ddd.
-O<level>	Précise le niveau d'optimisation souhaité pour la compilation. <level> vaut 1 2 ou 3. En général 2.
-I<chemin>	Ajoute <chemin> aux répertoires dans lesquels gcc ira chercher des fichiers headers .h. Exemple : <code>gcc -I/usr/local/PET/include -c monFichier.c</code> permet au compilateur de trouver des fichiers headers dans /usr/local/PET/include
-ansi	Indique que le <standard> (⇔la révision) du langage C utilisé est le C est la version « ainsi » (première version normalisée du C, datant de 1989).
-std=<standard>	Indique le nom du <standard> (⇔de la révision) du langage C à utiliser. Souvent <standard> sera c89 (norme « ansi » : le C de base) ou c99 (plus courant aujourd'hui).

### Options propre à l'étape d'édition des liens (-o)

-L<chemin>	Ajoute <chemin> aux répertoires dans lesquels gcc ira chercher des librairies binaires.
-l<nom_de_lib>	Indique qu'il faut lier la librairie <nom_de_lib> au binaire. Exemple : <code>gcc -o prg f1.o f2.o -L/usr/local/lib -lpet -lreadline</code> créera l'exécutable prg en liant file1.o, file2.o, et les binaires de la librairie « readline » libreadline.so et de la librairie « pet » libpet.so, en cherchant dans les chemins standards ansi que dans /usr/local/PET/lib

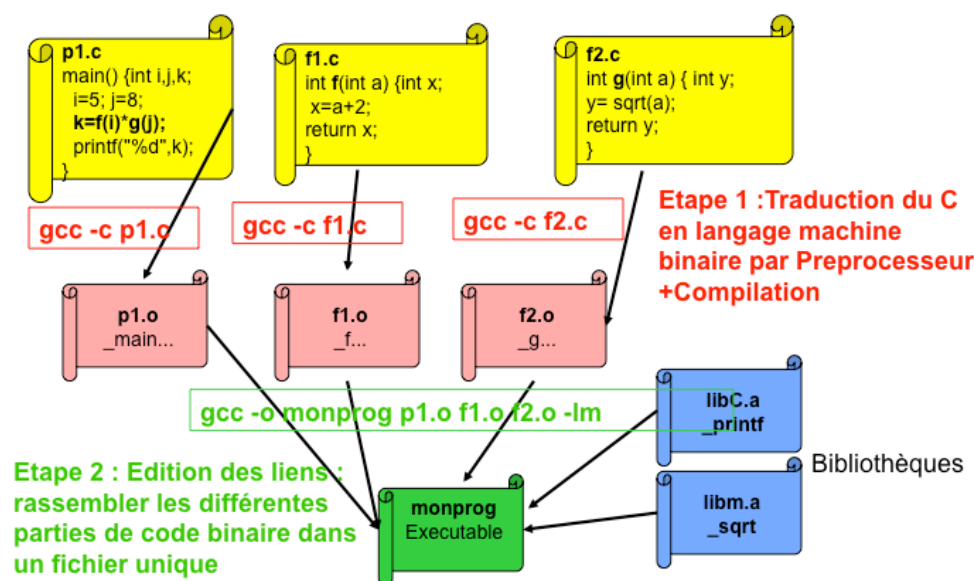
### 3. Exemple

On considère un programme réalisant le produit de  $(i+2)$  par la racine carré de  $j$ .

- La somme  $i+2$  est réalisée par une fonction  $f$ , qui est écrite dans le fichier `f1.c`
- La fonction  $g$ , écrite dans le fichier `f2.c`, se contente d'appeler la fonction racine carrée (`sqrt`) de la librairie mathématique du C.
- Le programme principal `main`, qui calcule  $f(i) * g(j)$ , se trouve dans le fichier `p1.c`

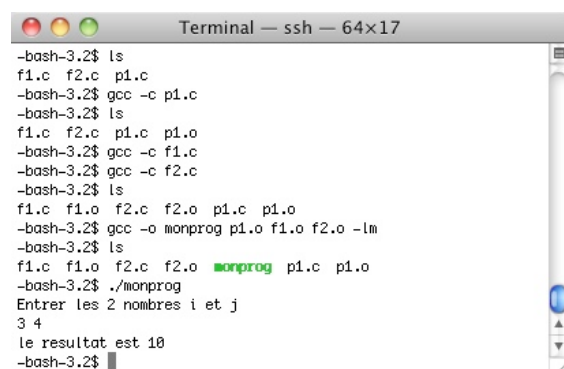
On veut construire le programme binaire exécutable qui regroupe tout cela.

2 étapes : compilation de chacun des fichiers source `.c` puis édition des liens.



Ainsi, pour créer le fichier binaire exécutable en mode « debug », il faut lancer successivement les commandes suivantes dans le Terminal :

```
gcc -c p1.c -g
gcc -c f1.c -g
gcc -c f2.c -g
gcc -o monprog p1.o f1.o f2.o -lm
```



```
Terminal — ssh — 64x17
-bash-3.2$ ls
f1.c f2.c p1.c
-bash-3.2$ gcc -c p1.c
-bash-3.2$ ls
f1.c f2.c p1.c p1.o
-bash-3.2$ gcc -c f1.c
-bash-3.2$ gcc -c f2.c
-bash-3.2$ ls
f1.c f1.o f2.c f2.o p1.c p1.o
-bash-3.2$ gcc -o monprog p1.o f1.o f2.o -lm
-bash-3.2$ ls
f1.c f1.o f2.c f2.o monprog p1.c p1.o
-bash-3.2$ ./monprog
Entrer les 2 nombres i et j
3 4
le resultat est 10
-bash-3.2$
```

Notez que le lien avec la librairie standard du C (`libc`) est sous-entendu.

L'utilitaire `make` et le fichier `Makefile` permettent de simplifier cette tâche.