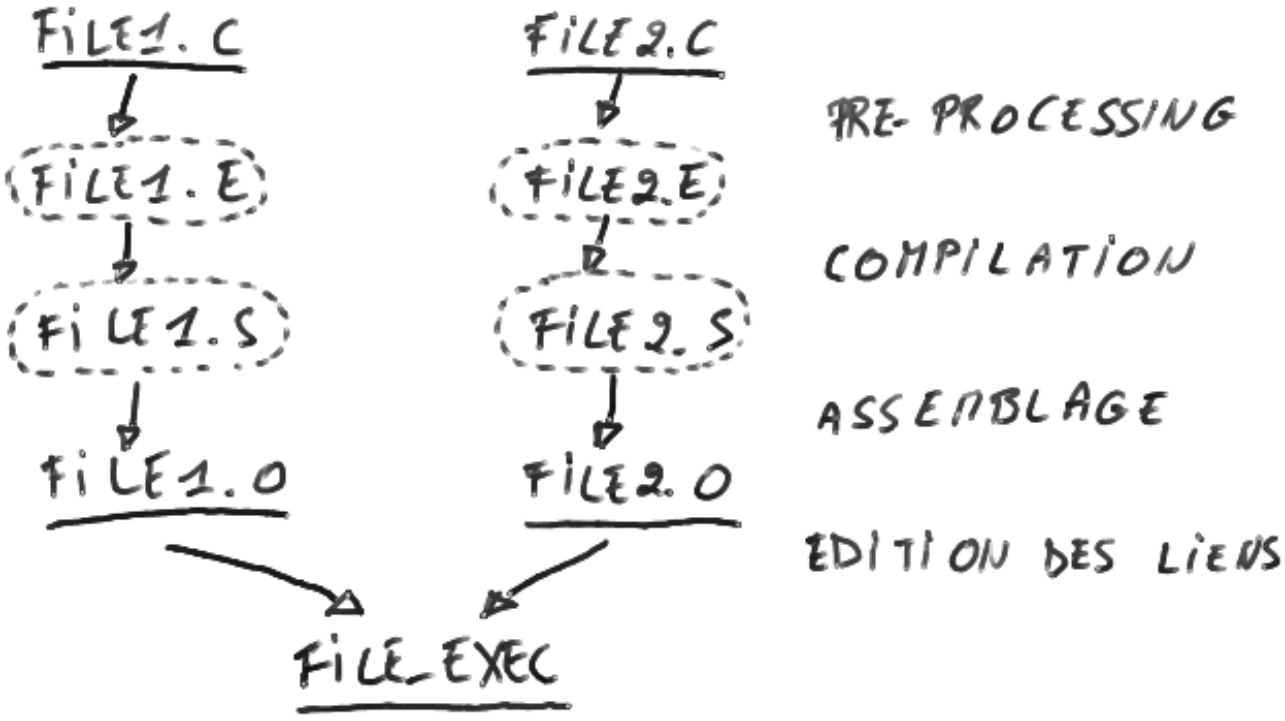
SINF1252
Résumé programmation C (partie 1)

Compilation avec GCC

Introduction

La compilation d'un programme C est un processus qui comprend plusieurs étapes (voir figure). On distingue en général la compilation et l'édition des liens. La compilation est le processus qui transforme un fichier source écrit en langage C en un fichier binaire appelé fichier objet. Un fichier objet contient la traduction de chaque fonction écrite en C dans du code en langage machine. Un fichier objet n'est pas directement exécutable. Les fichiers objet doivent être liés entre eux par l'étape d'édition des liens. De façon simplifiée, l'édition des liens consiste à générer un fichier exécutable en reliant entre eux les différents fichiers objets.

Par convention, les fichiers source écrits en langage C ont l'extension de fichier "**.c**", les fichiers objet ont l'extension "**.o**". En ce qui concerne les fichiers exécutables, la convention dépend de la plateforme. Sous linux, les fichiers exécutables n'ont en général pas d'extension.



Dans la figure ci-dessus, la compilation est divisée en plusieurs sous-étapes. La première étape est le pré-processing qui effectue une transformation du fichier source en un autre fichier C directement utilisable par le compilateur. La seconde étape est la compilation qui consiste à traduire le fichier C en un fichier en langage d'assemblage. Le langage d'assemblage

est dépendant du processeur de la machine. Ce fichier en langage d'assemblage est finalement transformé en code machine par un assembleur. Le code machine est placé dans un fichier objet[1].

Compilation d'un fichier source unique

La compilation d'un fichier C **file1.c** avec **gcc** peut se faire en une seule étape et générer un fichier exécutable **file_exec**.

```
gcc -o file_exec file1.c
```

La compilation peut également s'effectuer en plusieurs étapes. En général, la compilation et l'édition des liens sont séparées lorsqu'il y a plusieurs fichiers source à compiler et lier.

```
gcc -c -o file1.o file1.c
gcc -o file_exec file1.o
```

Compilation de multiples fichiers source

Lorsque le programme à générer comprend plusieurs fichiers source, il est également possible d'effectuer la compilation en une seule étape, en passant à **gcc** chacun des fichiers source en argument.

```
gcc -o file_exec file1.c file2.c
```

Lorsqu'il y a plusieurs fichiers source, on préfère séparer la compilation et l'édition des liens.

```
gcc -c -o file1.o file1.c
gcc -c -o file2.o file2.c
gcc -o file_exec file1.o file2.o
```

On sépare la compilation pour plusieurs raisons:

- si on modifie un seul fichier, il faut tout recompiler si on utilise la compilation en une étape. En séparant, on recompile uniquement les fichiers modifiés et on lie le tout. Cela peut réduire significativement le temps total de compilation.
- en général, la ligne de commande a une longueur limitée et la compilation en une seule étape est impossible.

Options importantes de GCC

Le tableau ci-dessous reprend quelques options importantes de GCC. Il est utile de spécifier systématiquement les options -Wall et -Werror car elles permettent de détecter facilement des erreurs de programmation en demandant au compilateur de renseigner tous les avertissements possibles. En pratique, les programmes que vous nous remettrez devront compiler sans erreurs avec ces options.

Option	Description	Exemple
-o *target*	Spécifie le nom du fichier de sortie (la cible).	gcc **-o** prog source.c
-c	Spécifie que seule l'étape de compilation doit être effectuée	gcc **-c** -o source.o source.c
-Wall	Spécifie que tous les avertissements doivent être montrés.	

1 Par défaut, les fichiers intermédiaires ("**.e**" et "**.s**") ne sont pas générés sur le disque mais sont créés en mémoire. Inutile de les chercher sur votre disque dur. Il est néanmoins possible de forcer gcc à les générer en utilisant les options **-E** et **-S** (voir la man page de gcc).

Option	Description	Exemple
-Werror	Spécifie que les avertissements sont considérés comme des erreurs (la compilation est empêchée dans ce cas).	
-I*chemin*	Spécifie un répertoire (chemin) additionnel où rechercher les fichiers header (avec l'extension "**.h**")	gcc -c **-Isrc/include** -o prog.o prog.c

Projet en plusieurs fichiers source

Petit exemple de programme écrit dans deux fichiers source séparés: **prog.c** et **util.c**. Le fichier **util.c** implémente une fonction **add()** qui fait l'addition de deux nombres entiers et retourne le résultat. Voici comment **util.c** implémente l'addition:

```
/* Contenu de util.c */
int add(int a, int b) {
  return a+b;
}
```

Voici comment on voudrait que **prog.c** utilise la fonction **add()** définie dans **util.c**.

```
/* Contenu de prog.c */
#include <stdio.h>
int main() {
  int sum= add(123, 589);
  printf("result: %d\n", sum);
  return 0;
}
```

Le problème est que si on compile **prog.c** tel quel, le compilateur va dire qu'il ne connaît pas la fonction **add()**[2]. Ce problème est résolu en écrivant un fichier d'en-tête (header) qui décrit les fonctions implémentées par **util.c**. Par convention, le fichier header a l'extension "**.h**" et a le même nom que le fichier source auquel il se rapporte. Nous le nommons donc **util.h**. Le fichier header **util.h** contient la signature de chaque fonction du fichier **util.c**. Voici à quoi **util.h** ressemble[3].

```
/* Contenu de util.h */
#ifndef __UTIL_H__
#define __UTIL_H__
int add(int a, int b);
#endif /* __UTIL_H__ */
```

Ce fichier est inclus dans le source **prog.c**, avec la directive **#include**. La version finale de **prog.c** sera donc la suivante:

---

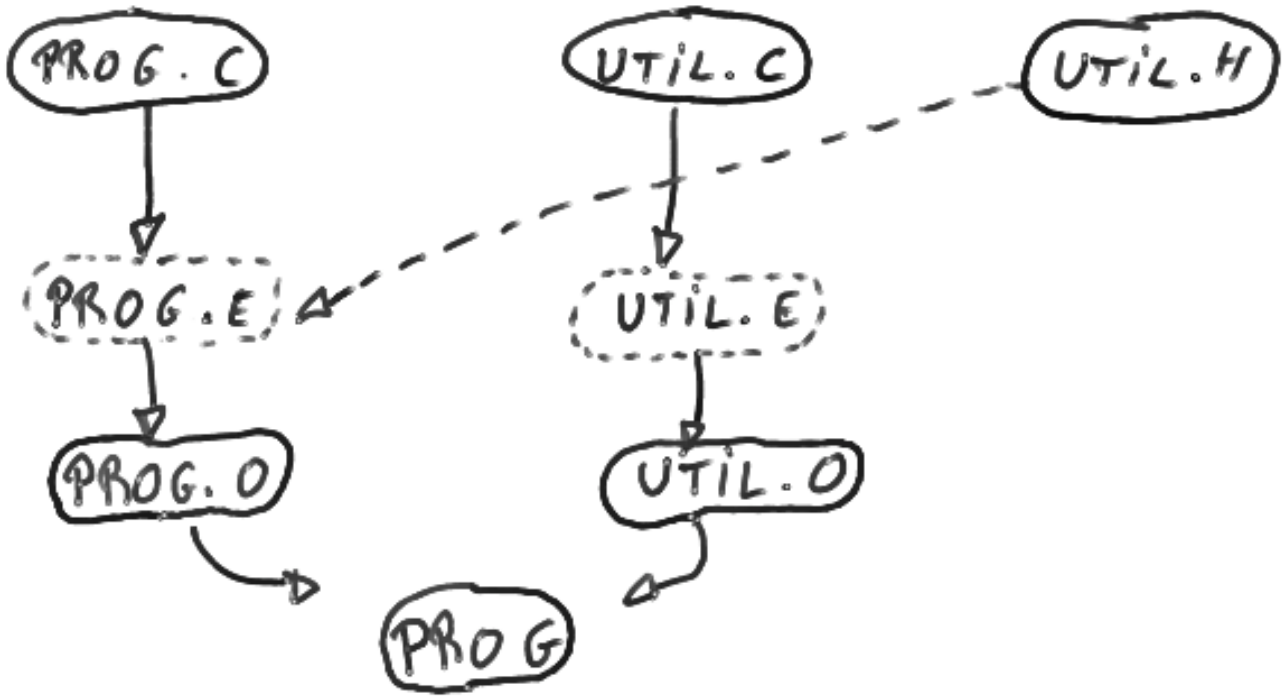
[2] Il faut parfois utiliser l'option -Wall pour que le compilateur affiche cette erreur.

[3] Les directives **#ifndef**, **#define** et **#endif** qui encadrent le contenu de **util.h** permettent de s'assurer que le fichier **util.h** ne sera inclus qu'une fois dans le fichier source qui l'utilise. Ces directives sont traitées par le pré-processeur. La directive **#define** définit un symbole nommé **__UTIL_H__**. Les directives **#ifndef** et **#endif** définissent un bloc de texte qui ne sera inclus par le préprocesseur que si le symbole n'est pas défini. Une règle de bonne pratique consiste à utiliser un symbole basé sur le nom du fichier header (ici, **__UTIL_H__** fait référence au fichier **util.h**).

```
/* Contenu de prog.c */
#include <stdio.h>
#include <util.h>
int main() {
  int sum= add(123, 589);
  printf("result: %d\n", sum);
  return 0;
}
```

La compilation de l'ensemble du projet est illustrée dans la figure suivante. C'est le rôle de l'étape de pré-processing de traiter les directives **#include** et de copier le contenu de **stdio.h** et **util.h** dans le fichier **prog.e** avant la compilation. En simplifiant un maximum, le contenu de prog.e généré par le compilateur ressemble au listing suivant (le contenu de **stdio.h** et **util.h** est copié dans le fichier source, à la place des directives **#include**).

```
/* Contenu de prog.e */
int printf(const char * restrict format, ...);
int add(int a, int b);
int main() {
  int sum= add(123, 589);
  printf("result: %d\n", sum);
  return 0;
}
```



Lors de la compilation de **prog.c**, il peut également être nécessaire d'indiquer où le fichier header **util.h** peut être trouvé. En effet, par défaut, le pré-processeur recherche les fichiers header dans une liste de répertoires standard (tels que /usr/include). Il est possible de mentionner des répertoires de recherche additionnels avec l'option **-I***chemin*. Dans notre cas, **util.h** se trouve dans le répertoire courant, il faudra donc compiler **prog.c** avec la ligne de commande suivante[4]:

```
gcc -c -Wall -Werror -I. -o prog.o prog.c
```

[4] Le répertoire courant sous unix est représenté par un point ("**.**").