Langage C avancé

François Cuvelier

Laboratoire d'Analyse Géométrie et Applications
Institut Galilée
Université Paris XIII.

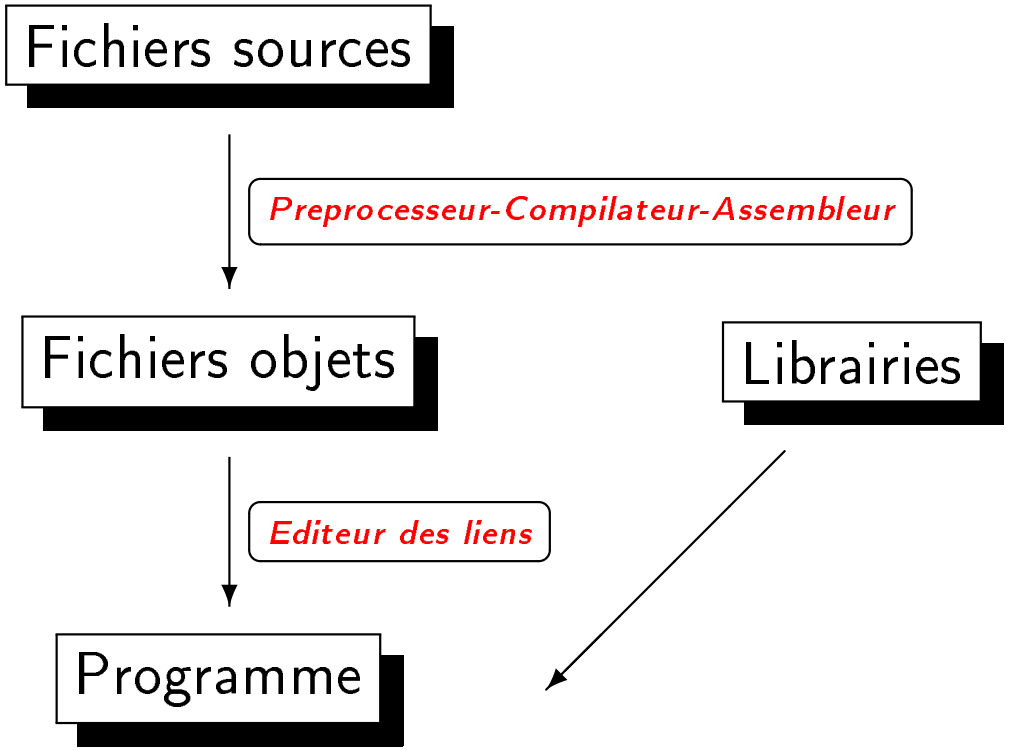
22 septembre 2010

Plan global

- Outils de programmation (UNIX)
  - Compilateur GCC
  - Création, gestion et utilisation de librairies
  - Makefile
  - Débuggeur GDB,
  - Interfaçage Matlab et C, Scilab C...
- Langage C évolué
  - opérateurs,
  - pointeurs,
  - structures,
  - allocation dynamique,
  - fonctions, ...

Outils de programmation - Plan

Schéma de compilation



Un exemple simple

Listing 1 – hello.c

```
1 #include <stdio.h>
2 int main() {
3     printf("Hello World!!!\n");
4     return 1;
5   }
```

▷ Comment créer le fichier objet *hello.o* ?

Un exemple simple

Listing 1 – hello.c

```c
#include <stdio.h>
int main() {
    printf("Hello World!!!\n");
    return 1;
  }
```

▷ Comment créer le fichier objet *hello.o* ?

```
# gcc -c hello.c
```

▷ Comment créer le programme executable *hello* ?

Un exemple simple

Listing 1 – hello.c

```
1 #include <stdio.h>
2 int main() {
3     printf("Hello World!!!\n");
4     return 1;
5   }
```

▷ Comment créer le fichier objet *hello.o* ?

```
# gcc -c hello.c
```

▷ Comment créer le programme executable *hello* ?

```
# gcc -o hello hello.c
```

ou

```
# gcc -c hello.c
# gcc -o hello hello.o
```

Compilation séparée

- *matrice.h matrice.c* : définition et fonctions pour les matrices.
- *vecteur.h vecteur.c* : définition et fonctions pour les vecteurs.
- *gauss.h gauss.c* : implémentation de la méthode de Gauss
- *prg2.c prg2.h* : programme principal

▷ Comment créer le programme executable *prg2* ?

Compilation séparée

- *matrice.h matrice.c* : définition et fonctions pour les matrices.
- *vecteur.h vecteur.c* : définition et fonctions pour les vecteurs.
- *gauss.h gauss.c* : implémentation de la méthode de Gauss
- *prg2.c prg2.h* : programme principal

▷ Comment créer le programme executable *prg2* ?

```
# gcc -c vecteur.c
# gcc -c matrice.c
# gcc -c gauss.c
# gcc -c prg2.c
# gcc -o prg2 prg2.o matrice.o vecteur.o gauss.o
```

Compilation séparée

- *matrice.h matrice.c* : définition et fonctions pour les matrices.
- *vecteur.h vecteur.c* : définition et fonctions pour les vecteurs.
- *gauss.h gauss.c* : implémentation de la méthode de Gauss
- *prg2.c prg2.h* : programme principal

▷ Comment créer le programme executable *prg2* ?

```
# gcc -c vecteur.c
# gcc -c matrice.c
# gcc -c gauss.c
# gcc -c prg2.c
# gcc -o prg2 prg2.o matrice.o vecteur.o gauss.o
```

Avantages de la compilation séparée

- Gain de temps
- Réutilisation
- Facilite le développement

Quelques options de GCC

- **Contrôle du type de sortie**
  - `-c` Compilation de fichiers sources sans édition des liens. (Création de fichiers objets)
  - `-o` *file* Place le résultat de la sortie dans le fichier *file*. (Création exécutable ou objet)
- **Editions des liens**
  - `-l`*nom* Utilise la librairie *libnom.a* lors de l'édition des liens.
- **Recherche de répertoire**
  - `-I`*dir* Ajoute le répertoire *dir* en tête de liste pour la recherche des fichiers *header* (*.h)
  - `-L`*dir* Ajoute le répertoire *dir* à la liste des répertoires pour rechercher les librairies.

▸ Lib.Stat. ▸ Lib.Share

Quelques options de GCC

- **preprocesseur**

  -D *macro valeur*

- **Debogage**

  -g Permet l'utilisation d'un logiciel de debogage (**gdb**, ...) à partir de l'exécutable du programme compilé.

- **Optimisation**

  -O1, -O2 ou -O3 l'optimisation est d'autant plus élevée que le nombre est grand.
  Par défaut : ? ?

Exercice (1/6)

prg4.c

```c
/* Fichier prog4.c */
#include <stdio.h>

void f1();
void f2();
void f3();
int main()
{
    /* Appel de la fonction f1 du fichier fic1.c */
   f1();
   /* Appel de la fonction f2 du fichier fic2.c */
   f2();
   /* Appel de la fonction f3 du fichier fic3.c */
   f3();
}
```

fic1.c

```c
/*Fichier fic1.c */
#include <stdio.h>
void f1()
{
   printf("Appel de f1\n");
}
```

fic2.c

```c
/*Fichier fic2.c */
#include <stdio.h>
void f2()
{
   printf("Appel de f2\n");
}
```

fic3.c

```c
/*Fichier fic3.c */
#include <stdio.h>
void f3()
{
   printf("Appel de f3\n");
}
```

Exercice

▷ Comment créer le programme exécutable *prg4* ?

▸ Avec make

Exercice

▷ Comment créer le programme exécutable *prg4* ?

Par exemple avec les commandes :

```
# gcc -c prg4.c
# gcc -c fic1.c
# gcc -c fic2.c
# gcc -c fic3.c
# gcc -o prg4 prg4.o fic1.o fic2.o fic3.o
```

▷ On modifie le fichier *fic2.c*. Comment créer le nouvel exécutable *prg4* ?

Par exemple avec les commandes :

```
# gcc -c fic2.c
# gcc -o prg4 prg4.o fic1.o fic2.o fic3.o
```

▸ Avec make

Exercice

▷ Que se passe-t-il lors de l'exécution des commandes suivantes ?

```
# gcc -c *.c
# gcc -o prg4 prg4.c
```

Exercice

▷ Que se passe-t-il lors de l'exécution des commandes suivantes ?

```
# gcc -c *.c
# gcc -o prg4 prg4.c
```

- **# gcc -c *.c**
  ⇒ *Compilation de tous les fichiers d'extension <.c> et création des fichiers objets correspondants.*
- **# gcc -o prg4 prg4.c**

```
# gcc -c *.c
```

```
# gcc -o prg4 prg4.c
/tmp/ccKoaxed.o: In function 'main':
/tmp/ccKoaxed.o(.text+0x11): undefined reference to 'f1'
/tmp/ccKoaxed.o(.text+0x16): undefined reference to 'f2'
/tmp/ccKoaxed.o(.text+0x1b): undefined reference to 'f3'  collect2: ld returned 1 exit status
```

Outils de programmation - Plan

Les librairies sous UNIX

Définition

Une librairie est ensemble cohérent de fonctions et sous-programmes **compilés**, «parfaitement» validés et testés, regroupés au sein d'une même archive.

exemple : la librairie MATH du langage C contient toutes les fonctions mathématiques usuelles. Elle est stockée sous forme de fichier : *libm.a* ou *libm.so* (habituellement répertoire */usr/lib*).

Deux types de librairies :

- librairies statiques (.a)
- librairies dynamiques (.so)

Librairies statiques

Le code objet d'une librairie nécessaire à l'exécution d'un programme est dupliqué dans le corps de l'exécutable lors de l'édition des liens.

- Avantages :
  Tout le code dont il a besoin pour fonctionner est inclu dans le fichier exécutable.
- Inconvénients :
  Mise à jour de la librairie $\Rightarrow$ recompilation du programme principal.
  Taille de l'exécutable.

Librairies dynamiques

L'exécutable ne contient qu'un lien vers cette librairie.

- Avantages :
  - Taille de l'exécutable,
  - Mise à jour de la librairie $\Rightarrow$ recompilation du programme principal INUTILE.
- Inconvénients :
  - Portabilité,
  - Le loader dynamique doit trouver la librairie (`LD_LIBRARY_PATH`)

Outils de gestion de librairies

- **ar** permet de gérer les librairies statiques.
  Options courantes :
  - **s** : écrire un index des fichiers objets contenu,
  - **c** : créer une archive,
  - **r** : insérer des fichiers avec remplacement,
  - **t** : liste le contenu de l'archive, ...
- **gcc -shared** permet de gérer les librairies dynamiques,
- **nm** affiche les symboles de fichiers objets ou de librairies.

Pour plus de renseignements **man** !

Exemple librairie : statique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie statique *libsample-static.a* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'executable *prg4-static* associé au fichier *prg4.c*.

▸ Options gcc

Exemple librairie : statique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie statique *libsample-static.a* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'executable *prg4-static* associé au fichier *prg4.c*.

1. Création des fichiers objets *fic1.o fic2.o fic3*
2. Création de la librairie statique *libsample-static.a*
3. Création de l'exécutable *prg4-static*

▸ Options gcc

Exemple librairie : statique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie statique *libsample-static.a* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'executable *prg4-static* associé au fichier *prg4.c*.

1. Création des fichiers objets *fic1.o fic2.o fic3*

```
# gcc -c fic1.c fic2.c fic3.c
```

2. Création de la librairie statique *libsample-static.a*

3. Création de l'exécutable *prg4-static*

▸ Options gcc

Exemple librairie : statique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie statique *libsample-static.a* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'executable *prg4-static* associé au fichier *prg4.c*.

1. Création des fichiers objets *fic1.o fic2.o fic3*

```
# gcc -c fic1.c fic2.c fic3.c
```

2. Création de la librairie statique *libsample-static.a*

```
# ar cr libsample-static.a fic1.o fic2.o fic3.o
```

3. Création de l'exécutable *prg4-static*

▸ Options gcc

Exemple librairie : statique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie statique *libsample-static.a* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'executable *prg4-static* associé au fichier *prg4.c*.

1. Création des fichiers objets *fic1.o fic2.o fic3*

```
# gcc -c fic1.c fic2.c fic3.c
```

2. Création de la librairie statique *libsample-static.a*

```
# ar cr libsample-static.a fic1.o fic2.o fic3.o
```

3. Création de l'exécutable *prg4-static*

```
# gcc -c prg4.c
# gcc -o prg4-static prg4.o -L. -lsample-static
```

▸ Options gcc

Exemple : librairie statique

▷ Le fichier *prg4.c* a été modifié. Que faut-il faire ?

Exemple : librairie statique

▷ Le fichier *prg4.c* a été modifié. Que faut-il faire ?

```
# gcc -c prg4.c
# gcc -o prg4-static prg4.o -L. -lsample-static
```

▷ Le fichier *fic2.c* a été modifié. Que faut-il faire ?

Exemple : librairie statique

▷ Le fichier *prg4.c* a été modifié. Que faut-il faire ?

```
# gcc -c prg4.c
# gcc -o prg4-static prg4.o -L. -lsample-static
```

▷ Le fichier *fic2.c* a été modifié. Que faut-il faire ?

```
# gcc -c fic2.c
# ar r libsample-static.a fic2.o
# gcc -o prg4-static prg4.o -L. -lsample-static
```

Exemple librairie : dynamique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie dynamique *libsample-share.so* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'exécutable *prg4-share* associé au fichier *prg4.c*.

▸ Options gcc

Exemple librairie : dynamique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie dynamique *libsample-share.so* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'exécutable *prg4-share* associé au fichier *prg4.c*.

1. Création des fichiers objets *fic1.o fic2.o fic3*
2. Création de la librairie dynamique *libsample-share.so*
3. Création de l'exécutable *prg4-share*

▸ Options gcc

Exemple librairie : dynamique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie dynamique *libsample-share.so* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'exécutable *prg4-share* associé au fichier *prg4.c*.

1. Création des fichiers objets *fic1.o fic2.o fic3*

```
# gcc -c fic1.c fic2.c fic3.c
```

2. Création de la librairie dynamique *libsample-share.so*

3. Création de l'exécutable *prg4-share*

▸ Options gcc

Exemple librairie : dynamique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie dynamique *libsample-share.so* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'exécutable *prg4-share* associé au fichier *prg4.c*.

1. Création des fichiers objets *fic1.o fic2.o fic3*

```
# gcc -c fic1.c fic2.c fic3.c
```

2. Création de la librairie dynamique *libsample-share.so*

```
# gcc -shared -o libsample-share.so fic1.o fic2.o fic3.o
```

3. Création de l'exécutable *prg4-share*

▸ Options gcc

Exemple librairie : dynamique

A partir des sources de l'exemple précédant ▸ Sources, on veux créer une librairie dynamique *libsample-share.so* «contenant» les fonctions f1, f2 et f3. Cette librairie sera alors utilisée pour créer l'exécutable *prg4-share* associé au fichier *prg4.c*.

1. Création des fichiers objets *fic1.o fic2.o fic3*

```
# gcc -c fic1.c fic2.c fic3.c
```

2. Création de la librairie dynamique *libsample-share.so*

```
# gcc -shared -o libsample-share.so fic1.o fic2.o fic3.o
```

3. Création de l'exécutable *prg4-share*

```
# gcc -c prg4.c
# gcc -o prg4-share prg4.o -L. -lsample-share
```

▸ Options gcc

Exemple : librairie dynamique

▷ Le fichier *prg4.c* a été modifié. Que faut-il faire ?

Exemple : librairie dynamique

▷ Le fichier *prg4.c* a été modifié. Que faut-il faire ?

```
# gcc -c prg4.c
# gcc -o prg4-shared prg4.o -L. -lsample-share
```

▷ Le fichier *fic2.c* a été modifié. Que faut-il faire ?

Exemple : librairie dynamique

▷ Le fichier *prg4.c* a été modifié. Que faut-il faire ?

```
# gcc -c prg4.c
# gcc -o prg4-shared prg4.o -L. -lsample-share
```

▷ Le fichier *fic2.c* a été modifié. Que faut-il faire ?

```
# gcc -c fic2.c
# gcc -shared -o libsample-share.so fic1.o fic2.o fic3.o
```

Exemple : gestion de librairies

```
# ar cr libsimple.a fic1.o fic3.o
```

Exemple : gestion de librairies

```
# ar cr libsimple.a fic1.o fic3.o Insertion (avec remplacement)
# ar t libsimple.a
```

Exemple : gestion de librairies

```
# ar cr libsimple.a fic1.o fic3.o  Insertion (avec remplacement)
# ar t libsimple.a  Affiche le contenu de l'archive
fic1.o

fic3.o
# ar a libsimple.a fic2.o
```

Exemple : gestion de librairies

```
# ar cr libsimple.a fic1.o fic3.o  Insertion (avec remplacement)
# ar t libsimple.a  Affiche le contenu de l'archive
fic1.o

fic3.o
# ar a libsimple.a fic2.o  Ajoute fic2.o
# ar t libsimple.a
```

Exemple : gestion de librairies

```
# ar cr libsimple.a fic1.o fic3.o  Insertion (avec remplacement)
# ar t libsimple.a  Affiche le contenu de l'archive
fic1.o

fic3.o
# ar a libsimple.a fic2.o  Ajoute fic2.o
# ar t libsimple.a
fic1.o

fic3.o

fic2.o
```

Exemple : gestion de librairies

```
# ar d libsimple.a fic3.o
```

Exemple : gestion de librairies

```
# ar d libsimple.a fic3.o   Détruit fic3.o
# ar t libsimple.a
```

Exemple : gestion de librairies

```
# ar d libsimple.a fic3.o   Détruit fic3.o
# ar t libsimple.a
fic1.o

fic2.o
# ar a libsimple.a fic3.o
```

Exemple : gestion de librairies

```
# ar d libsimple.a fic3.o   Détruit fic3.o
# ar t libsimple.a

fic1.o

fic2.o
# ar a libsimple.a fic3.o   Ajouter fic3.o à la fin
# ar t libsimple.a
```

Exemple : gestion de librairies

```
# ar d libsimple.a fic3.o   Détruit fic3.o
# ar t libsimple.a
fic1.o

fic2.o
# ar a libsimple.a fic3.o   Ajouter fic3.o à la fin
# ar t libsimple.a
fic1.o

fic2.o

fic3.o
```

Exemple : gestion de librairies

 # nm libsimple.a

```
fic1.o:

00000000 T f1

U _GLOBAL_OFFSET_TABLE_

U printf

fic2.o:

00000000 T f2

U printf

fic3.o:

...
```

Exemple : gestion de librairies

```
# nm -s libsimple.a
Archive index:
f1 in fic1.o
f2 in fic2.o
f3 in fic3.o

fic1.o:
00000000 T f1
         U puts
fic2.o:
00000000 T f2
         U puts
fic3.o:
00000000 T f3
         U puts
```

Fichiers headers et librairies

simple.h

```c
/* Fichier simple.h associe a la librairie libsimple.a */
#ifndef _SIMPLE_H
#define _SIMPLE_H

void f1(); /* Affiche "Appel de f1" */
void f2(); /* Affiche "Appel de f2" */
void f3(); /* Affiche "Appel de f3" */
#endif
```

prg5.c

```c
/* Fichier prog5.c */
#include "simple.h"

int main()
{
   f1(); /* Appel de la fonction f1 (libsimple.a) */
   f2(); /* Appel de la fonction f2 (libsimple.a) */
   f3(); /* Appel de la fonction f3 (libsimple.a) */
}
```

Outils de programmation - Plan

Make

Présentation

Cet utilitaire permet (entre autres) la mise à jour automatique d'exécutables, de librairies, ... Et ceci en «minimisant» le nombre de tâches à accomplir.

Principe de fonctionnement

L'utilitaire make utilise un fichier de description (makefile) permettant de «fabriquer» des cibles. La première cible est la cible par défaut.

```
cible␣:␣dependances
_____Regles␣de␣production
```

Tabulation

Le trait avant la *Règles de production* représente le caractère <tabulation>. Il est OBLIGATOIRE.

- Vérification des dates de création des dépendances.
- (Re)Création des dépendances dont les dates de création se révèlent antérieures à celles des sources correspondants.
- Vérification de la date de création de la cible.
- (Re)Création de la cible si sa mise à jour est nécessaire.

Exemple - Dépendances

Nous allons établir les dépendances permettant d'écrire un makefile ayant pour cible principale l'exécutable *prg4* associé au programme *prg4.c*.

▸ Sources

cible	dépendances
prg4	

Exemple - Dépendances

Nous allons établir les dépendances permettant d'écrire un makefile ayant pour cible principale l'exécutable *prg4* associé au programme *prg4.c*.

▸ Sources

cible	dépendances
prg4	prg4.c fic1.c fic2.c fic3.c

Exemple - Dépendances

Nous allons établir les dépendances permettant d'écrire un makefile ayant pour cible principale l'exécutable *prg4* associé au programme *prg4.c*.

▸ Sources

cible	dépendances
prg4	prg4.c fic1.c fic2.c fic3.c

makefile

```
1 # Mon premier makefile
2
3 prg4 : prg4.o fic1.o fic2.o fic3.o
4 ________gcc -c fic1.c
5 ________gcc -c fic2.c
6 ________gcc -c fic3.c
7 ________gcc prg4.o fic1.o fic2.o fic3.o -o prg4
```

makefile1

```
1 # Mon premier makefile
2
3 prg4 : prg4.c fic1.c fic2.c fic3.c
4 ________gcc prg4.c fic1.c fic2.c fic3.c -o prg4
```

Voici le résultat de la commande unix **ls**

```
# ls
fic1.c fic2.c fic3.c makefile prg4.c
```

▷ Comment créer le programme exécutable *prg4* avec *make* ?

▸ Avec gcc

Voici le résultat de la commande unix **ls**

```
# ls
fic1.c fic2.c fic3.c makefile prg4.c
```

▷ Comment créer le programme exécutable *prg4* avec *make* ?

```
# make
gcc -c fic1.c

gcc -c fic2.c

gcc -c fic3.c

gcc -o prg4 prg4.o fic1.o fic2.o fic3.o
```

▷ On modifie le fichier *fic2.c*. Comment créer le nouvel exécutable *prg4* avec *make* ?

▸ Avec gcc

Voici le résultat de la commande unix **ls**

```
# ls
fic1.c fic2.c fic3.c makefile prg4.c
```

▷ Comment créer le programme exécutable *prg4* avec *make* ?

```
# make
gcc -c fic1.c

gcc -c fic2.c

gcc -c fic3.c

gcc -o prg4 prg4.o fic1.o fic2.o fic3.o
```

▷ On modifie le fichier *fic2.c*. Comment créer le nouvel exécutable *prg4* avec *make* ?

Affichage identique au précédent

▸ Avec gcc

Voici le résultat de la commande unix **ls**

```
# ls
fic1.c fic2.c fic3.c makefile prg4.c
```

▷ Comment créer le programme exécutable *prg4* avec *make* ?

```
# make
gcc -c fic1.c
gcc -c fic2.c
gcc -c fic3.c
gcc -o prg4 prg4.o fic1.o fic2.o fic3.o
```

▷ On modifie le fichier *fic2.c*. Comment créer le nouvel exécutable *prg4* avec *make* ?

Affichage identique au précédent

Remarque

La modification d'un des fichiers source entraine la compilation de l'ensemble des fichiers !

Ce *makefile* est loin d'être «optimum»

▸ Avec gcc

Exemple - Dépendances optimales

Nous allons établir les dépendances «optimales» permettant d'écrire un makefile ayant pour cible principale l'exécutable *prg4* associé au programme *prg4.c*. ▸ Sources

cible	dépendances
prg4	
prg4.o	
fic1.o	
fic2.o	
fic3.o	

Exemple - Dépendances optimales

Nous allons établir les dépendances «optimales» permettant d'écrire un makefile ayant pour cible principale l'exécutable *prg4* associé au programme *prg4.c*. ▸ Sources

cible	dépendances
prg4	prg4.o fic1.o fic2.o fic3.o
prg4.o	
fic1.o	
fic2.o	
fic3.o	

Exemple - Dépendances optimales

Nous allons établir les dépendances «optimales» permettant d'écrire un makefile ayant pour cible principale l'exécutable *prg4* associé au programme *prg4.c*. ▸ Sources

cible	dépendances
prg4	prg4.o fic1.o fic2.o fic3.o
prg4.o	prg4.c
fic1.o	
fic2.o	
fic3.o	

Exemple - Dépendances optimales

Nous allons établir les dépendances «optimales» permettant d'écrire un makefile ayant pour cible principale l'exécutable *prg4* associé au programme *prg4.c*. ▸ Sources

cible	dépendances
prg4	prg4.o fic1.o fic2.o fic3.o
prg4.o	prg4.c
fic1.o	fic1.c
fic2.o	fic2.c
fic3.o	fic3.c

Exemple - Dépendances optimales

Nous allons établir les dépendances «optimales» permettant d'écrire un makefile ayant pour cible principale l'exécutable *prg4* associé au programme *prg4.c*. ▸ Sources

cible	dépendances
prg4	prg4.o fic1.o fic2.o fic3.o
prg4.o	prg4.c
fic1.o	fic1.c
fic2.o	fic2.c
fic3.o	fic3.c

makefile2

```
1  # Mon premier makefile <<optimum>>
2
3  prg4 : prg4.o fic1.o fic2.o fic3.o
4  ________gcc -o prg4 prg4.o fic1.o fic2.o fic3.o
5
6  prg4.o : prg4.c
7  ________gcc -c prg4.c
8
9  fic1.o : fic1.c
10 ________gcc -c fic1.c
11
12 fic2.o : fic2.c
13 ________gcc -c fic2.c
14
15 fic3.o : fic3.c
16 ________gcc -c fic3.c
```

Voici le résultat de la commande unix **ls**

```
# ls
fic1.c fic2.c fic3.c makefile makefile2 prg4.c
```

▷ Comment créer le programme exécutable *prg4* avec *make* et le fichier *makefile2* ?

Voici le résultat de la commande unix **ls**

```
# ls
fic1.c fic2.c fic3.c makefile makefile2 prg4.c
```

▷ Comment créer le programme exécutable *prg4* avec *make* et le fichier *makefile2* ?

```
# make -f makefile2
gcc -c fic1.c
gcc -c fic2.c
gcc -c fic3.c
gcc -o prg4 prg4.o fic1.o fic2.o fic3.o
```

▷ On modifie le fichier *fic2.c*. Comment créer le nouvel exécutable *prg4* avec *makefile2* ?

```
# make -f makefile2
gcc -c fic2.c
gcc -o prg4 prg4.o fic1.o fic2.o fic3.o
```

▷ Que fait la commande UNIX **ls -l** ?

▷ Que fait la commande UNIX **ls -l** ?

Elle affiche le détail (option -l) du contenu du répertoire courant

```
# ls -l
-rw-r-r- 1 cuvelier cuvelier 80 2008-09-04 15:28 fic1.c
-rw-r-r- 1 cuvelier cuvelier 832 2008-09-05 07:21 fic1.o
-rw-r-r- 1 cuvelier cuvelier 83 2008-09-05 13:44 fic2.c
-rw-r-r- 1 cuvelier cuvelier 832 2008-09-05 13:45 fic2.o
-rw-r-r- 1 cuvelier cuvelier 82 2008-09-04 15:28 fic3.c
-rw-r-r- 1 cuvelier cuvelier 832 2008-09-05 07:21 fic3.o
-rw-r-r- 1 cuvelier cuvelier 152 2008-09-05 13:25 makefile
-rw-r-r- 1 cuvelier cuvelier 254 2008-09-05 13:40 makefile2
-rwxr-xr-x 1 cuvelier cuvelier 6455 2008-09-05 13:46 prg4*
-rw-r-r- 1 cuvelier cuvelier 301 2003-03-28 12:36 prg4.c
-rw-r-r- 1 cuvelier cuvelier 828 2008-09-05 07:21 prg4.o
```

▷ Que fait alors la commande **make -f makefile2** ?

▷ Que fait la commande UNIX **ls -l** ?

Elle affiche le détail (option -l) du contenu du répertoire courant

```
# ls -l
-rw-r-r- 1 cuvelier cuvelier 80 2008-09-04 15:28 fic1.c
-rw-r-r- 1 cuvelier cuvelier 832 2008-09-05 07:21 fic1.o
-rw-r-r- 1 cuvelier cuvelier 83 2008-09-05 13:44 fic2.c
-rw-r-r- 1 cuvelier cuvelier 832 2008-09-05 13:45 fic2.o
-rw-r-r- 1 cuvelier cuvelier 82 2008-09-04 15:28 fic3.c
-rw-r-r- 1 cuvelier cuvelier 832 2008-09-05 07:21 fic3.o
-rw-r-r- 1 cuvelier cuvelier 152 2008-09-05 13:25 makefile
-rw-r-r- 1 cuvelier cuvelier 254 2008-09-05 13:40 makefile2
-rwxr-xr-x 1 cuvelier cuvelier 6455 2008-09-05 13:46 prg4*
-rw-r-r- 1 cuvelier cuvelier 301 2003-03-28 12:36 prg4.c
-rw-r-r- 1 cuvelier cuvelier 828 2008-09-05 07:21 prg4.o
```

▷ Que fait alors la commande **make -f makefile2** ?

Rien !

```
# make -f makefile2
make: « prg4 » est à jour.
```

Exemple : Macro-définitions

Makefile avec macro-définitions

```
1  # EXEMPLE : MACRO-DEFINITIONS
2  #=========================
3  # Utilisation de variables definies par l'utilisateur
4  #
5
6  COMPILATEUR = gcc
7  OPTIONS     = -O
8  OBJETS      = prg4.o fic1.o fic2.o fic3.o
9  PROGRAMME   = prg4
10
11 $(PROGRAMME) : $(OBJETS)
12 ----------$(COMPILATEUR) $(OPTIONS) $(OBJETS) -o $(PROGRAMME)
13
14 prg4.o : prg4.c
15 ----------$(COMPILATEUR) $(OPTIONS) -c prg4.c
16
17
18 fic1.o : fic1.c
19 ----------$(COMPILATEUR) $(OPTIONS) fic1.c
20
21 fic2.o : fic2.c
22 ----------$(COMPILATEUR) $(OPTIONS) fic2.c
23
24 fic3.o : fic3.c
25 ----------$(COMPILATEUR) $(OPTIONS) fic3.c
```

Macro-définitions internes

Il existe des macro-définitions internes modifiables ou non.

Macro-définitions internes **non modifiables**

Macro	Signification
$*	nom du fichier, sans extension
$<	nom du fichier, avec extension
$?	Liste des dépendances plus récentes que la cible
$@	nom complet du fichier cible

Macro-définitions internes

Il existe des macro-définitions internes modifiables ou non.

Macro-définitions internes **modifiables**

Macro	Signification
`CC`	Compilateur C
`CPP`	Préprocesseur
`CXX`	Compilateur C++
`F77`	Compilateur fortran
`LD`	Editeur de liens
`AR`	Archivage (librairies)
`CFLAGS`	Options du compilateur C
`CPPFLAGS`	Options du préprocesseur
`CXXFLAGS`	Options du compilateur C++
`F77FLAGS`	Options du compilateur fortran
`LDFLAGS`	Options de l'éditeur des liens
...	

Règles par défaut

Elles permettent la description implicite de la construction de cibles.

Makefile : Règles par défaut

```
1 # EXEMPLE 3
2 # =========
3 # Utilisation des regles internes de derivation
4 #
5
6 OBJETS = prg4.o fic1.o fic2.o fic3.o
7 PROGRAMME = prg4
8
9 $(PROGRAMME) : $(OBJETS)
```

```
# make
cc -c -o prg4.o prg4.c

cc -c -o fic1.o fic1.c

cc -c -o fic2.o fic2.c

cc -c -o fic3.o fic3.c

cc prg4.o fic1.o fic2.o fic3.o -o prg4
```

Macro-définitions internes

Makefile : Modification des regles internes de derivation

```
1  # EXEMPLE 4
2  # ==========
3  # Modification des regles internes de derivation
4  #
5  CC      = gcc
6  CFLAGS  = -g
7  OBJETS  = prg4.o fic1.o fic2.o fic3.o
8  PROGRAMME = prg4
9
10 $(PROGRAMME) : $(OBJETS)
```

```
# make
gcc -g -c -o prg4.o prg4.c

gcc -g -c -o fic1.o fic1.c

gcc -g -c -o fic2.o fic2.c

gcc -g -c -o fic3.o fic3.c

gcc prg4.o fic1.o fic2.o fic3.o -o prg4
```

Quelques options de Make

-f *filename* Utilise le fichier *filename* comme fichier de description.

-p Liste des règles de dérivation internes et des dépendances prédéfinies.

-s Exécute mais n'affiche pas les commandes des règles de dérivations.

-n Affiche mais n'exécute pas les commandes des règles de dérivations.

Exemple

Le répertoire courant contient les fichiers (main) *prg1.c* et *prg2.c* ainsi que les fichiers *f1.c f2.c f3.c*. Nous voulons construire les exécutables *prg1* et *prg2* dépendants respectivement de *prg1.o f1.o f2.o f3.o* et *prg2.o f1.o f2.o f3.o*. Les deux programmes utilisent la librairie <MATH> et on souhaite pouvoir debugguer les codes.

Exemple

Le répertoire courant contient les fichiers (main) *prg1.c* et *prg2.c* ainsi que les fichiers *f1.c f2.c f3.c*. Nous voulons construire les exécutables *prg1* et *prg2* dépendants respectivement de *prg1.o f1.o f2.o f3.o* et *prg2.o f1.o f2.o f3.o*. Les deux programmes utilisent la librairie <MATH> et on souhaite pouvoir debugguer les codes.
Voici un exemple de makefile permettant de gérer les deux programmes.

Makefile

```
1  # EXERCICE
2
3  CC      = gcc
4  CFLAGS = -g
5  LDFLAGS = -lm
6
7  all : prg1 prg2
8
9  prg1 : prg1.o f1.o f2.o f3.o
10
11 prg2 : prg2.o f1.o f2.o f3.o
12
13 clean :
14         rm -f *.o prg1 prg2
```

Exemple

▷ Que fait la commande *make clean* ?

Makefile

```
1  # EXERCICE
2
3  CC      = gcc
4  CFLAGS = -g
5  LDFLAGS = -lm
6
7  all : prg1 prg2
8
9  prg1 : prg1.o f1.o f2.o f3.o
10
11 prg2 : prg2.o f1.o f2.o f3.o
12
13 clean :
14         rm -f *.o prg1 prg2
```

Exemple

Makefile

```
1  # EXERCICE
2
3  CC      = gcc
4  CFLAGS = -g
5  LDFLAGS = -lm
6
7  all : prg1 prg2
8
9  prg1 : prg1.o f1.o f2.o f3.o
10
11 prg2 : prg2.o f1.o f2.o f3.o
12
13 clean :
14         rm -f *.o prg1 prg2
```

▷ Que fait la commande *make clean* ?

Réalise la cible **clean**.

```
# make clean

rm -f *.o prg1 prg2
```

▷ Que fait la commande *make* ?

Exemple

Makefile

```
1  # EXERCICE
2
3  CC      = gcc
4  CFLAGS = -g
5  LDFLAGS = -lm
6
7  all : prg1 prg2
8
9  prg1 : prg1.o f1.o f2.o f3.o
10
11 prg2 : prg2.o f1.o f2.o f3.o
12
13 clean :
14         rm -f *.o prg1 prg2
```

▷ Que fait la commande *make clean* ?

Réalise la cible **clean**.

```
# make clean
rm -f *.o prg1 prg2
```

▷ Que fait la commande *make* ?

Réalise la cible par défaut (la première : **all**).

```
# make
gcc -g -c -o prg1.o prg1.c

gcc -g -c -o f1.o f1.c

gcc -g -c -o f2.o f2.c

gcc -g -c -o f3.o f3.c

gcc -lm prg1.o f1.o f2.o f3.o -o prg1

gcc -g -c -o prg2.o prg2.c

gcc -lm prg2.o f1.o f2.o f3.o -o prg2
```