Gestion Automatisée de la Compilation

`Makefile` et `make`

Makefile

- Outils permet d'automatiser le processus de compilation
- Fonctionne avec un système de dépendances
- Ne recompile que les fichiers qui nécéssite d'être recompilé
- Peut être utilisé pour plusieurs langages

Exemple d'utilisation

- Un programmeur écrit un programme composé de :
  - Le fichier m1.c qui contient le main
  - Les fichiers m2.c et m2.h qui contiennent des fonctions utiles
  - Les fichiers Liste.c et Liste.c qui contiennent la définition de la structure liste

Exemple (suite)

- Le programmeur travaille sur les fichiers .c
- En particulier il travaille sur m2.c
- Les fichier m1 et Liste ne sont pas modifiés pour le moment.

Processus de compilation : 2 possibilités

- Version longue :
  - gcc -c m1.c
  - gcc -c m2.c
  - gcc -c Liste.c
  - gcc -o m1 Liste.o m1.o m2.o
- Version courte :
  - Gcc -o m1 m1.c m2.c Liste.c

Problème de l'approche précédente :

- Dans le cas de la version courte
  - Recompilation de tous les fichiers
  - Seul m2.c a changé
  - → perte de temps
- Dans le cas de la version longue :
  - Recompilation de m1.c, m2.c et Liste.c
  - Puis édition de liens (gcc -o m1 *.o)
  - → perte de temps (m1 et Liste recompilé pour rien)

Solution : Makefile

- En fonction de la date de modification des fichiers
- Et des règles de dépendances
- Makefile peut déterminer quels fichiers re-compiler
  - Si dateModif(fichier.c)> dateModif(fichier.o) Alors recompiler fichier.c

Description du fichier

- all : nom_fichier1 nom_fichier2
  - Les noms des programmes à créer
- Règle de dépendance et construction :
- fichier.o : fichier.c fichier.h
  [Tabulation] Commande compilation
- Signifie que fichier.o dépend de fichier.c et fichier.h (si l'un des deux a été modifié après la modification de fichier.o on va recompiler)
-

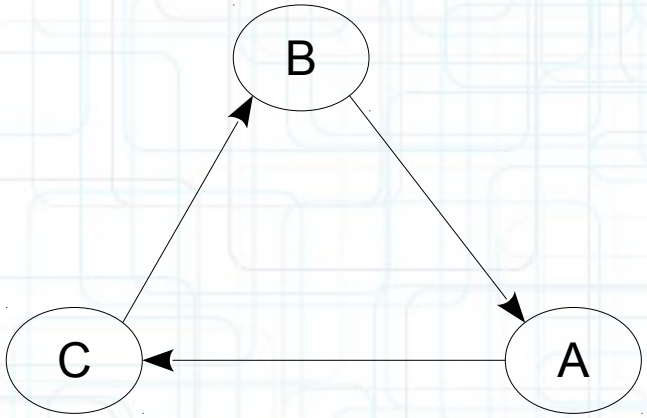
Makefile version simple

```
all: m1

m1 : m1.o m2.o Liste.o
	gcc -o m1 m1.o m2.o Liste.o
m1.o: m1.c
	gcc -c m1.c

m2.o: m2.c m2.h
	gcc -c m2.c

Liste.o : Liste.c Liste.h
	gcc -c Liste.c

clean:
```

- All : programme à obtnir
- Programme m1 dépend de m1.o et m2.o
- …
- Clean permet de supprimer les fichiers .o et l'exécutable.

Structures des dépendances

- Pour que le processus termine on ne peut pas avoir de « circuits »
- A.o : B.o
  - …
- B.o : C.o
  - …
- C.o : A.o
  - …



Makefile : Invocation

- Makefile est indépendant de C
- Il fonctionne avec un programme appellé make
  - `make`
  - `make clean`
  - `make m1`
  - `make Liste.o`

Makefile : version évoluée

```
CC=gcc
FLAGS= -Wall
OBJETS= m1.o m2.o Liste.o

all: m1
m1: $(OBJETS)
$(CC) $(FLAGS) -o  $@ $(OBJETS)

m1.o: m1.c
   $(CC) $(FLAGS) -c $<

m2.o: m2.c  m2.h
   $(CC) $(FLAGS) -c $<
Liste.o : Liste.c Liste.h
   $(CC) $(FLAGS) -c $<
```

Makefile : Syntaxe

- On peut définir des variables
  - CC=gcc
  - CC est le compilateur
  - $(CC)
  - On récupère la valeur de CC
- $@ et $<
  - $@ est la partie gauche d'uné dépendance
    - X.o : fichier.c fichier.h
    - $@ vaut X.o
  - $< est le premier argument de la dépendance :
    - $< vaut fichier.c

Makefile : version compacte

```
CC=gcc
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=m1.c m2.c Liste.c
OBJECTS=$(SOURCES:.c=.o)
EXECUTABLE=m1
.PHONY : clean
all: $(SOURCES)  $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
   $(CC) $(LDFLAGS) $(OBJECTS) -o $@
.c.o:
   $(CC) $(CFLAGS) $< -o $@
clean :
   rm -f $(OBJETS) $(EXECUTABLE)
```