



# Chap. 8 Modularisation

**I2181A**  
**Langage C :**  
**modularisation**

Anthony Legrand

# La découpe en module

Module =

- une boîte à outils comme  
une interface java

et

- une classe qui implémente  
cette interface

# Définir un module

- ▶ Un fichier entête (extension .h)  
≈ une interface en Java
- ▶ Un fichier source (extension .c)  
≈ une classe qui implémente cette interface

# Les fichiers d'entête

Un **header** comprend toutes les informations **publiques** du module

- ▶ définitions de constantes (`#define`)
- ▶ définitions de types (`typedef`)
- ▶ prototypes de fonctions
- ▶ spécifications des fonctions (`PRE`, `POST`, `RES`)
- ▶ **Attention: pas de code!**

# Exemple de header

```
#ifndef _PILE_H_
#define _PILE_H_

/* définition de constantes */
#define OK 1

/* définition de types */
typedef ... Pile;

/* déclaration de fonctions */
Pile init ();
int pop (Pile* p);
void push (Pile* p, int val);

#endif // _PILE_H_
```

} + specs

# Eviter la double inclusion

Des directives conditionnelles du préprocesseur permettent d'éviter une double inclusion de modules (lors de l'édition de liens)

```
#ifndef _PILE_H_
#define _PILE_H_

...

#endif // _PILE_H_
```

# Les fichiers sources: inclusions

```
/* fichier pile.c */
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include "pile.h"
```

} Librairies standard

} Header dans le  
répertoire courant

# Les fichiers sources: implémentation

```
Pile init () {  
    ...  
}  
  
int pop (Pile* p) {  
    ...  
}  
  
void push (Pile* p, int val) {  
    ...  
}
```



# Construction d'une application

9

La génération d'une application se fait en deux étapes :

## 1. **compilation**

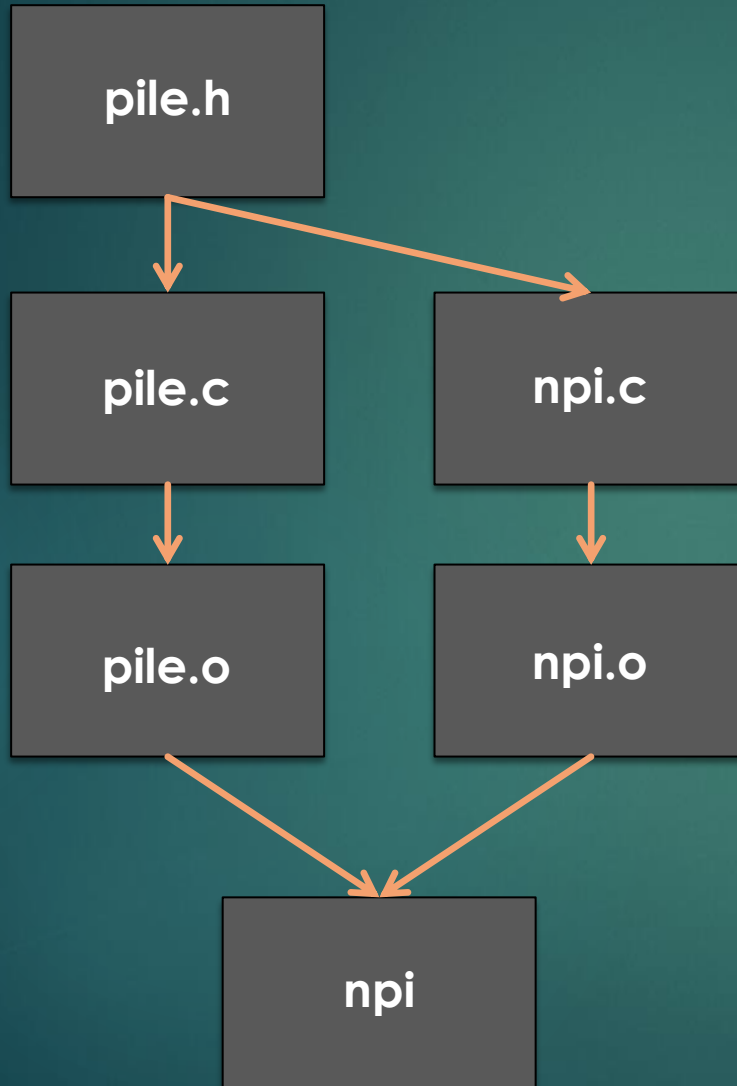
après avoir effectué la précompilation, le compilateur produit un fichier objet .o pour chaque fichier source .c qu'il compile

## 2. **édition des liens (*linkage*)**

produit un exécutable en assemblant les fichiers objets .o

# Construction d'une application

10



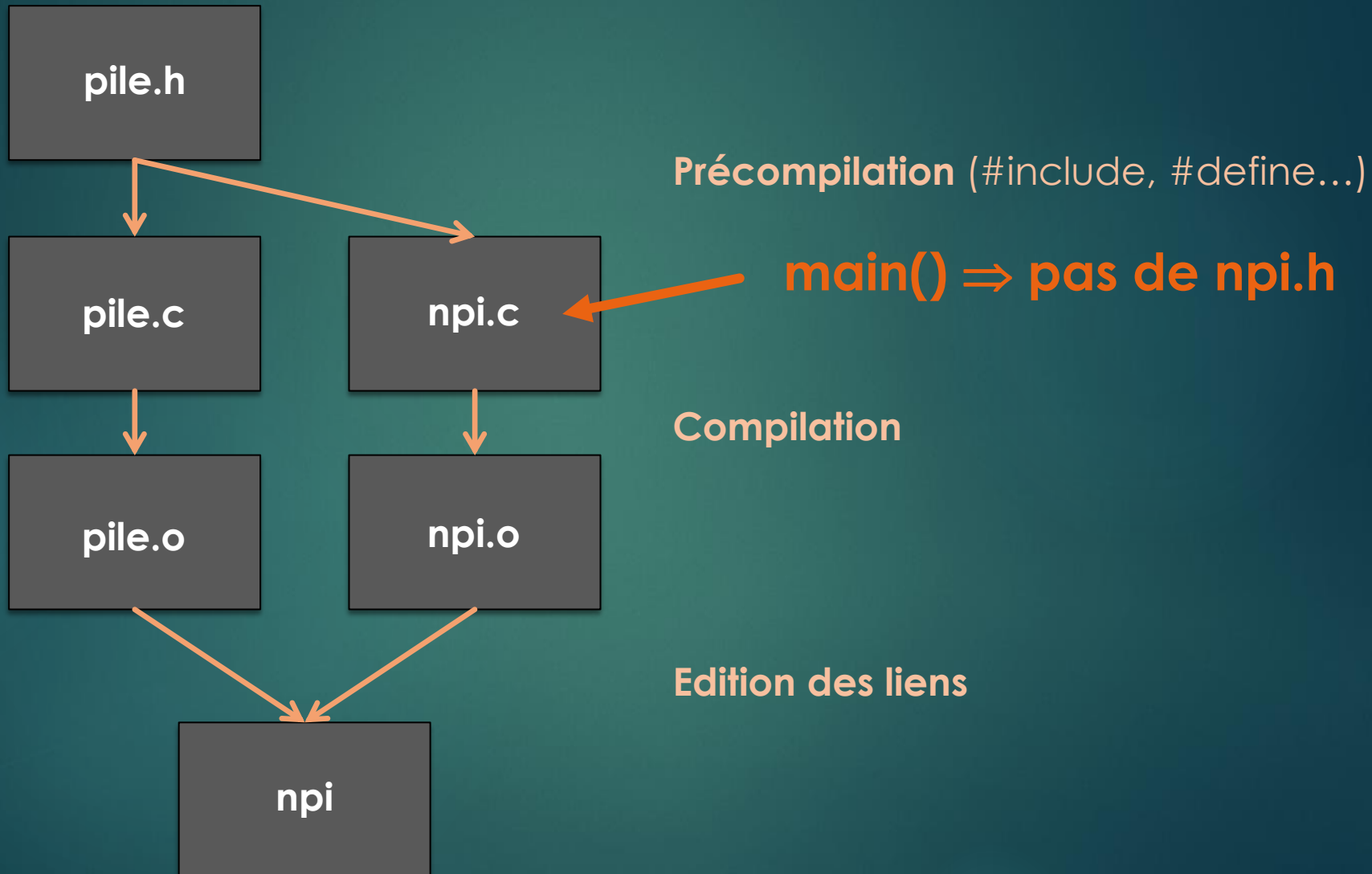
**Précompilation** (`#include`, `#define`...)

**Compilation**

**Edition des liens**

# Construction d'une application

11



# Makefile : Objectifs

12

- ▶ Un *makefile* permet de ne construire que ce qui n'est plus « à jour ».
- ▶ La compilation séparée permet de mieux organiser ses projets et de développer des bibliothèques de fonctions.
- ▶ Utile pour le développement de grosses applications.

# Makefile

13

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
npi : npi.o pile.o  
    cc $(CFLAGS) -o npi npi.o pile.o
```

```
npi.o : npi.c pile.h  
    cc $(CFLAGS) -c npi.c
```

```
pile.o : pile.c pile.h  
    cc $(CFLAGS) -c pile.c
```

```
clean :  
    rm *.o  
    rm npi
```

# Makefile

14

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
npi : npi.o pile.o  
    cc $(CFLAGS) -o npi npi.o pile.o
```

```
npi.o : npi.c pile.h  
    cc $(CFLAGS) -c npi.c
```

```
pile.o : pile.c pile.h  
    cc $(CFLAGS) -c pile.c
```

```
clean :  
    rm *.o  
    rm npi
```



options de compilation

# Makefile

15

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
npi : npi.o pile.o  
    cc $(CFLAGS) -o npi npi.o pile.o
```

```
npi.o : npi.c pile.h  
    cc $(CFLAGS) -c npi.c
```

```
pile.o : pile.c pile.h  
    cc $(CFLAGS) -c pile.c
```

```
clean :  
    rm *.o  
    rm npi
```

**règle d'édition de liens  
(pour générer l'exécutable *npi*)**



# Makefile

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
npi : npi.o pile.o  
    cc $(CFLAGS) -o npi npi.o pile.o
```

```
npi.o : npi.c pile.h  
    cc $(CFLAGS) -c npi.c  
  
pile.o : pile.c pile.h  
    cc $(CFLAGS) -c pile.c
```

```
clean :  
    rm *.o  
    rm npi
```



**règles de compilation  
(pour générer les fichiers objet)**



# Makefile

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
npi : npi.o pile.o  
    cc $(CFLAGS) -o npi npi.o pile.o
```

```
npi.o : npi.c pile.h  
    cc $(CFLAGS) -c npi.c
```

```
pile.o : pile.c pile.h  
    cc $(CFLAGS) -c pile.c
```

```
clean :  
    rm *.o  
    rm npi
```

→ règle utilitaire  
(pour effacer les  
fichiers générés)

# Makefile

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
np_i : np_i.o pile.o
```

```
cc $(CFLAGS) -o np_i np_i.o pile.o
```

```
np_i.o : np_i.c pile.h
```

```
cc $(CFLAGS) -c np_i.c
```

```
pile.o : pile.c pile.h
```

```
cc $(CFLAGS) -c pile.c
```

```
clean :
```

```
rm *.o
```

```
rm np_i
```



cible (target)

# Makefile

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
mpi : mpi.o pile.o
```

```
cc $(CFLAGS) -o mpi mpi.o pile.o
```

```
mpi.o : mpi.c pile.h
```

```
cc $(CFLAGS) -c mpi.c
```

```
pile.o : pile.c pile.h
```

```
cc $(CFLAGS) -c pile.c
```

```
clean :
```

```
rm *.o
```

```
rm mpi
```



**dépendances**

# Makefile

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
npi : npi.o pile.o
```

```
cc $(CFLAGS) -o npi npi.o pile.o
```

```
npi.o : npi.c pile.h
```

```
cc $(CFLAGS) -c npi.c
```

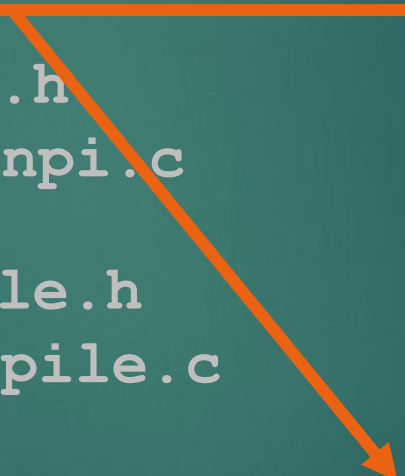
```
pile.o : pile.c pile.h
```

```
cc $(CFLAGS) -c pile.c
```

```
clean :
```

```
rm *.o
```

```
rm npi
```



**Commande exécutée si  
dépendance(s) plus récente(s)  
que cible ou cible non existante**

# Makefile

```
CFLAGS=-std=c17 -pedantic -Wvla -Werror -Wall
```

```
npi : npi.o pile.o
```

```
cc $(CFLAGS) -o npi npi.o pile.o
```

```
npi.o : npi.c pile.h
```

```
cc $(CFLAGS) -c npi.c
```

```
pile.o : pile.c pile.h
```

```
cc $(CFLAGS) -c pile.c
```

```
clean :
```

```
rm *.o
```

```
rm npi
```

**Tabulation obligatoire!**

Liste de règles qui ont

- ▶ une condition de dépendance (optionnelle)

```
npi : npi.o pile.o
```

- ▶ une action (optionnelle)

```
cc $(CFLAGS) -o npi npi.o pile.o
```

# Commande make

- ▶ **make**

mise à jour de la première cible  
du makefile : **npi**

- ▶ **make npi.o**

mise à jour de la cible **npi.o**

- ▶ **make clean**

provoque l'effacement des modules  
objets et de l'exécutable → à exécuter  
avant publication/partage du code ou  
après modification du *makefile* lui-même

# Makefile avec règle 'all'

```
CFLAGS=-std=c11 -pedantic -Wvla -Werror -Wall
```

```
all : np1 tree
```

```
tree : tree.o pile.o  
    cc $(CFLAGS) -o tree tree.o pile.o
```

```
np1 : np1.o pile.o  
    cc $(CFLAGS) -o np1 np1.o pile.o
```

```
...
```

► **make all** ou **make**

génère/actualise tous les exécutables



- Plus d'infos sur make et makefile :

<https://linuxpedia.fr/doku.php/dev/makefile>

# Différents types de modules

26

Distinction entre des modules :

- **bibliothèque**

fichiers .h et .c mais pas de main()

➔ « boîte à outils »

- **configuration**

uniquement header .h

➔ fichier de config, définissant des constantes, des macros et/ou des types

- **application**

uniquement fichier source .c avec main()

➔ fichier permettant de générer un programme exécutable