

Chaine de compilation

Compilateur
Warnings
Compilation séparée
Préprocesseur
Makefile
Edition de liens et bibliothèques

000

Chaine de compilation

```
int main()
{
    int a=3;
    a++;
    int b=a+2;

    char texte[]="j aime l informatique";

    return 0;
}
```

Code (C)
(=langage humain)

Compilateur

Fichier executable
(=binaire)

(Windows: .exe
Linux: nom quelconque)

```
b8af 0860 0055 482d a808 6000 4883 f80e
4889 e577 025d c3b8 0000 0000 4885 c07f
f45d bfa8 0860 00ff e00f 1f80 0000 0000
b8a8 0860 0055 482d a808 6000 48c1 f803
4889 e548 89c2 48c1 ea3f 4801 d048 89c6
48d1 fe75 025d c3ba 0000 0000 4885 d274
f45d bfa8 0860 00ff e20f 1f80 0000 0000
803d 4104 2000 0075 1155 4889 e5e8 7eff
ffff 5dc6 052e 0420 0001 f3c3 0f1f 4000
4883 3d08 0220 0000 741b b800 0000 0048
85c0 7411 55bf 9006 6000 4889 e5ff d05d
e97b ffff ffe9 76ff ffff 9090 5548 89e5
c745 fc03 0000 0083 45fc 018b 45fc 83c0
0289 45f8 c745 e06a 2061 69c7 45e4 d6d5
206c c745 e820 69e6 66c7 45e6 6f72 d6d1
c745 f074 6971 7566 c745 f465 00b8 0000
0000 5dc3 9090 9090 9090 9090 9090 9090
4889 6c24 d84c 8964 24e0 488d 2d77 0120
004c 8d25 6801 2000 4889 5c24 d04c 896c
24e8 4c89 7424 f04c 897c 24f8 4883 ec38
4c29 e541 89ff 4989 f648 c1fd 0349 89d5
31db e829 feff ff48 85ed 741a 0f1f 4000
4c89 ea4c 89f6 4489 ff41 ff14 dc48 83c3
0148 39eb 75ea 488b 5c24 0848 8b6c 2410
4c8b 6424 184c 8b6c 2420 4c8b 7424 284c
```

001

Compilateur

gcc



<http://gcc.gnu.org/>
(C, C++, Obj-C, Fortran, Ada, Go)

GNU Compiler Collection

Le + répandu sous Linux

D'autres existent:

Nwcc
Clang
Quick C (Microsoft)
Turbo C (Embarcadero)
XL C (IBM)

Note:
Compilateur C++ compatibles
BorlandC++
Intel C++ Compiler
Visual Studio
Turbo C++
ProDev
Solaris Studio

002

Compilateur

Remarque:

Ne pas confondre **IDE** et **Compilateur**

éditer du texte/code

génère le binaire

Emacs
QtCreator
MS Visual C++
Dev C++
Eclipse
Code::Blocks
Anjuta
KDevelop



ceci n'est pas
un compilateur!

003

Compilateur

En pratique



004

Compilateur: Paramètres de GCC

gcc possède des paramètres:

- Warnings
- Debug
- Optimisation
- Chemins d'accès
- Lien avec librairies
- Constantes
- ...

beaucoup d'options!

<http://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html>

005

Chaine de compilation

→ **Compilateur**
→ **Warnings**
Compilation séparée
Préprocesseur
Makefile
Edition de liens et librairies

006

Compilateur

Ex: Warning

```
#include <stdio.h>

int main()
{
    float *a;
    printf("%f", *a);

    return 0;
}
```

sans option de Warning

```
[damien@garonne ~/work/2012_2013_teaching/2012_3eti_projet_c/cours/src/code]$ gcc compilation.c
[damien@garonne ~/work/2012_2013_teaching/2012_3eti_projet_c/cours/src/code]$ gcc compilation.c -Wuninitialized
compilation.c: In function 'main':
compilation.c:7:17: warning: 'a' is used uninitialized in this function [-Wuninitialized]
```

avec option de Warning

en francais:
compilation.c, ligne 17. Warning 'a' est utilisé sans être initialisé dans cette fonction.

Warning = Aide pour le programmeur
= Lisible pour le programmeur
= Evite les erreurs "silencieuses"

007

Warnings typiques:

```
int oublie_retour(int valeur)
{
    int valeur_a_retourner=0;
    valeur_a_retourner=valeur+1;
}

int* recupere_adresse_temporaire(int valeur)
{
    return &valeur;
}

int main()
{
    int a=oublie_retour(3);
    printf("%d\n",a);

    int tableau[2]={1,2,3};

    int b=3;
    int *p=recupere_adresse_temporaire(b);
    printf("%d\n",*p);

    int somme=0;
    unsigned int k=0;
    for(k=5;k>=0;--k)
        somme += 3;

    return 0;
}
```

Sans warnings

\$ gcc mon_fichier.c

compile

Fonctionne?

008

Warnings typiques:

```
int oublie_retour(int valeur)
{
    int valeur_a_retourner=0;
    valeur_a_retourner=valeur+1;
}

int* recupere_adresse_temporaire(int valeur)
{
    return &valeur;
}

int main()
{
    int a=oublie_retour(3);
    printf("%d\n",a);

    int tableau[2]={1,2,3};

    int b=3;
    int *p=recupere_adresse_temporaire(b);
    printf("%d\n",*p);

    int somme=0;
    unsigned int k=0;
    for(k=5;k>=0;--k)
        somme += 3;

    return 0;
}
```

pas de retour

adresse temporaire

dépassement tableau

toujours vrai: boucle infinie

Ajout de warnings

\$ gcc mon_fichier.c -Wall -Wextra

```
mon_fichier.c: In function 'oublie_retour':
mon_fichier.c:6:9: warning: variable 'valeur_a_retourner' set but not used [-Wunused-but-set-variable]
mon_fichier.c: In function 'recupere_adresse_temporaire':
mon_fichier.c:12:5: warning: function returns address of local variable [enabled by default]
mon_fichier.c: In function 'main':
mon_fichier.c:20:5: warning: excess elements in array initializer [enabled by default]
mon_fichier.c:20:5: warning: (near initialization for 'tableau') [enabled by default]
mon_fichier.c:20:9: warning: unused variable 'tableau' [-Wunused-variable]
mon_fichier.c:28:5: warning: comparison of unsigned expression >= 0 is always true [-Wtype-limits]
mon_fichier.c: In function 'oublie_retour':
mon_fichier.c:8:1: warning: control reaches end of non-void function [-Wreturn-type]
```

009

Warnings : Bonnes pratiques



Toujours activer un maximum de Warnings

-Wall -Wextra

Indispensable!!

conteneurs de multiples warnings

=> Toujours compiler avec `$gcc -Wall -Wextra`

D'autres Warnings très utiles:

(non compris dans -Wall ni -Wextra)

-Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum
-Wwrite-strings -Wpointer-arith -Wcast-qual -Wredundant-decls
-Winit-self

010

Warnings : Bonnes pratiques

Toujours activer un maximum de Warnings

Ne **jamais** se priver du travail du compilateur!

Si après analyse des Warnings inutiles gachent la **lisibilité**
(ex. unused variables)

annule ce Warning spécifique

Option: `-Wno-<nom_warning>`

ex. `-Wall -Wextra -Wno-unused-variable`

011

Warnings : Bonnes pratiques

```
gcc -Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings  
-Wpointer-arith -Wcast-qual -Wredundant-decls -Winit-self
```

Trop long à écrire ?

1 unique fois dans un fichier
utilisez les scripts

script = lignes de commandes écrites dans un fichier
exécutées les unes derrière les autres

ex.



<nom_script>.sh

```
#!/bin/bash  
  
<mes_lignes_de_scripts>  
<...>
```

012

Script pour compiler

En pratique:

1. créez le fichier:



mon_script.sh

dans le même repertoire que:



mon_programme.c

2. y inscrire:

```
#!/bin/bash  
gcc mon_fichier.c -Wall -Wextra -o mon_executable
```

3. rendez le fichier executable: \$ chmod +x mon_script.sh

4. lancez le: \$./mon_script.sh

013

Script pour compiler

En pratique:

Avec l'ensemble des paramètres de compilation:

```
#!/bin/bash  
CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings"  
gcc mon_fichier.c -o mon_executable ${CFLAGS}
```

déclaration d'une variable (+ lisible)

utilisation d'une variable

Note: En anglais: options de compilation = *flags*
options de compilation du compilateur C = *Cflags*

version générique:

```
#!/bin/bash  
CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings"  
nom_de_fichier=mon_fichier.c  
echo "Je compile" ${nom_de_fichier}  
gcc ${nom_de_fichier} -o mon_executable ${CFLAGS}
```

014

Options de debug



-g

Toujours compiler avec l'option de Debug
lors du développement

Permet l'utilisation des debuggers (gdb, kdbg, ddd, valgrind, ...)

Pour résumer, CFLAGS minimales à toujours utiliser:

```
-Wall -Wextra -g
```

015

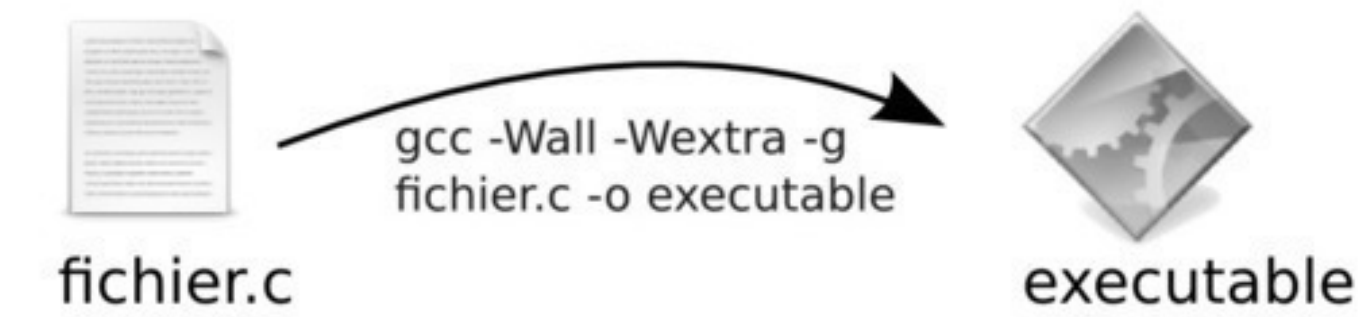
Chaine de compilation

Compilateur
Warnings
→ **Compilation séparée**
Préprocesseur
Makefile
Edition de liens et librairies

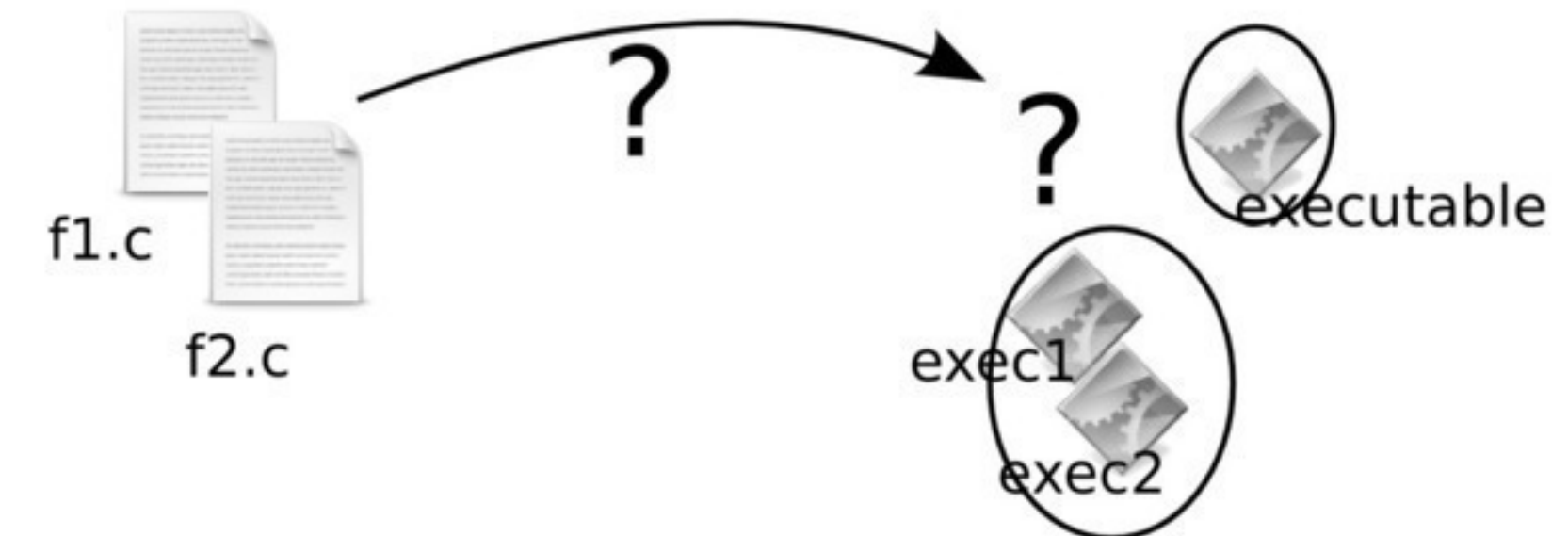
016

Compilation séparée

1 Fichier:



2 Fichiers:

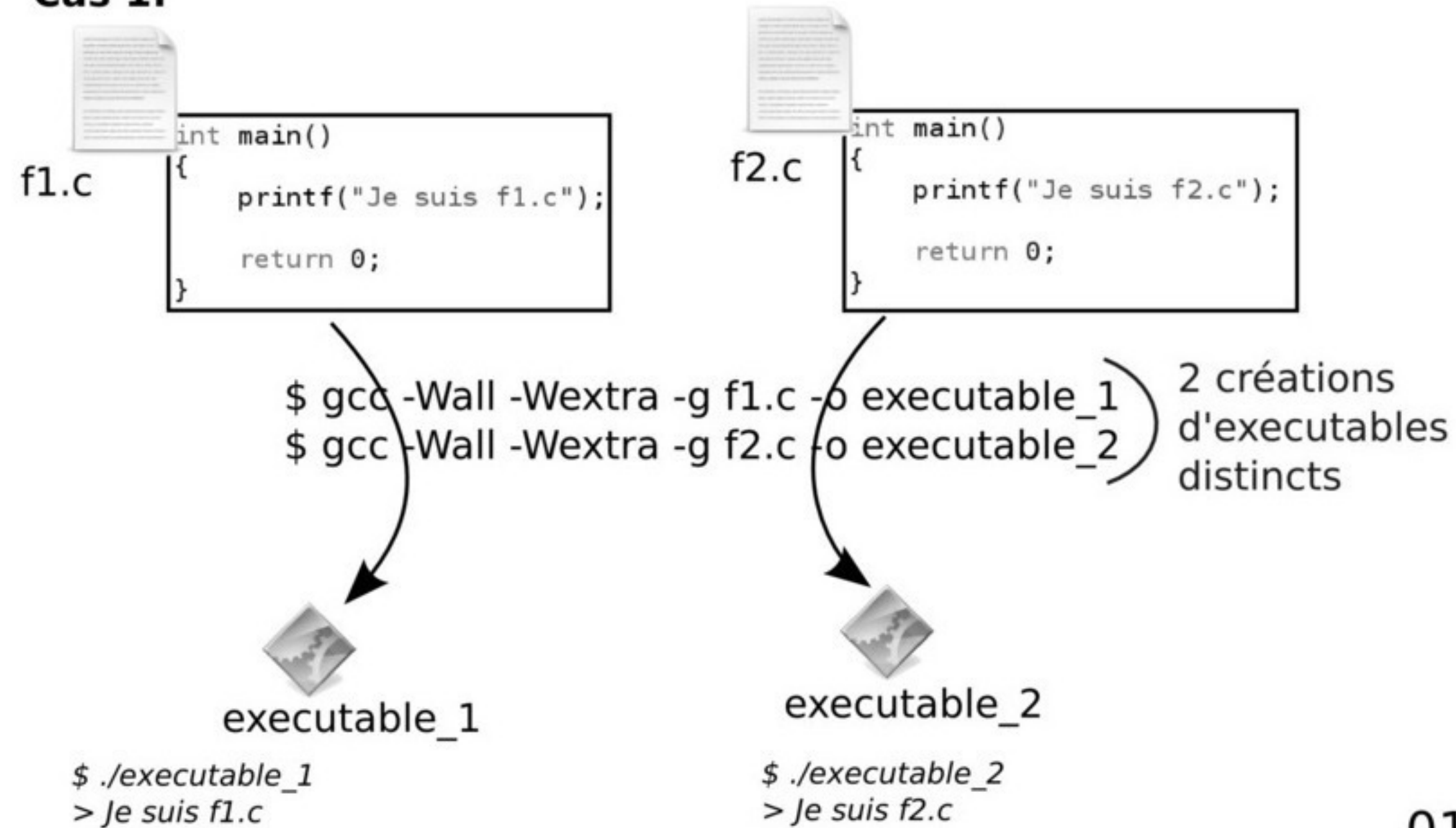


017

Cas de 2 fichiers

Un executable => 1 point d'entrée (int main())

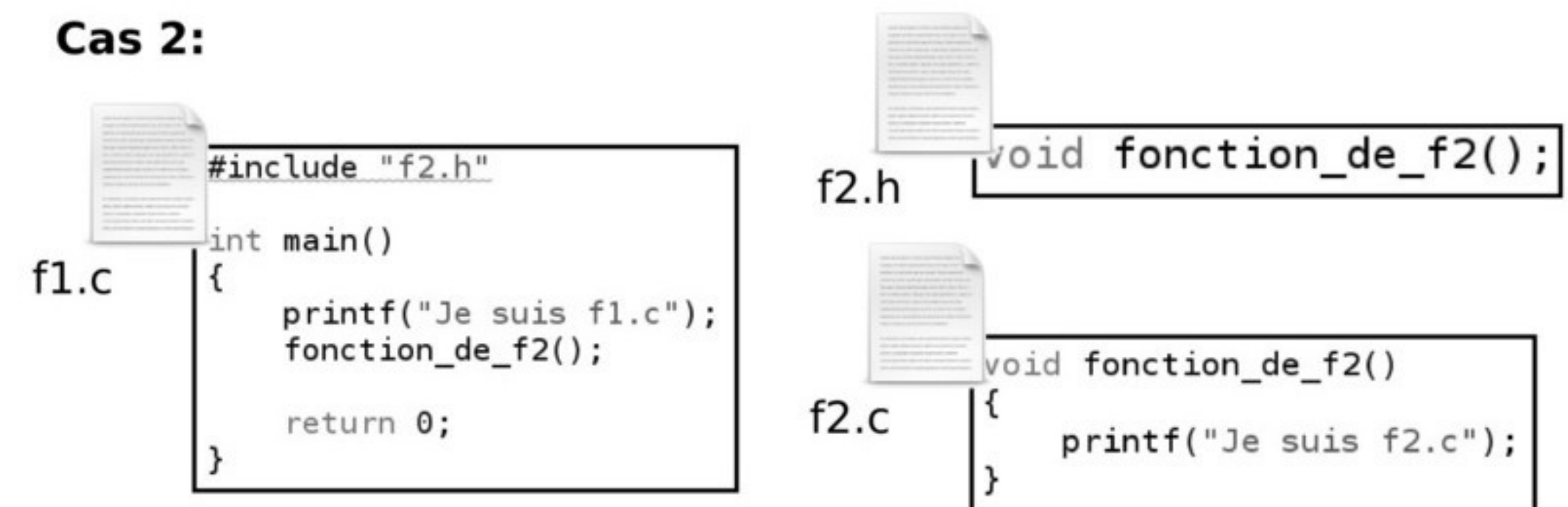
Cas 1:



018

Cas de 2 fichiers

Cas 2:



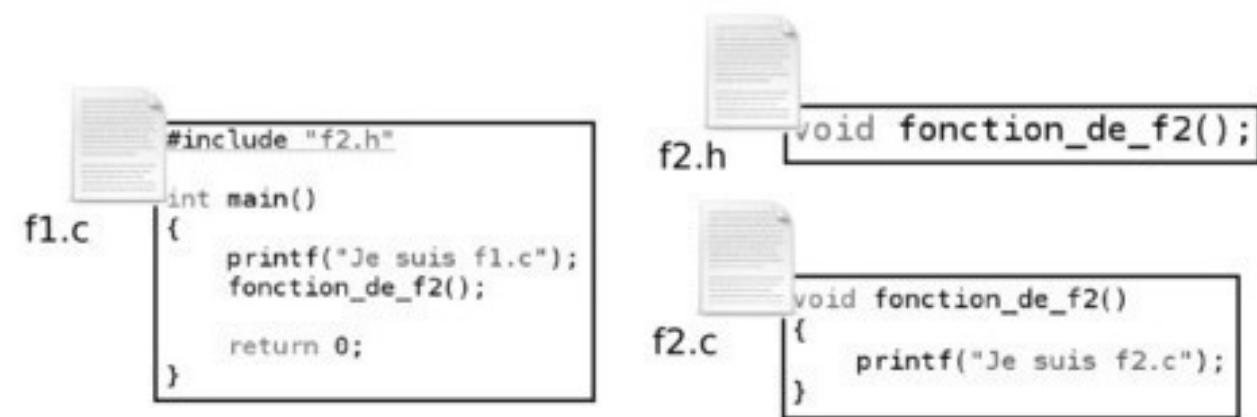
f2 ne contient pas de fonction: *int main()*
=> pas d'executable associé à f2 uniquement!

f2 agit en tant que "librairie" pour le code appelé dans f1

019

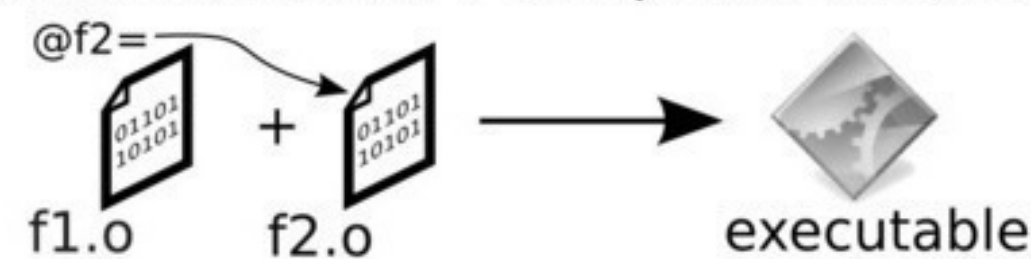
Cas de 2 fichiers: compilation séparée

Cas 2:



Principe:

- On compile tout ce qu'on peut de f1
=> cad: tout sauf la fonction de f2 (adresse à définir)
- On compile tout ce qu'on peut de f2
=> cad: tout
- On rassemble les 2 binaires + complète l'adresse de fonction définitive

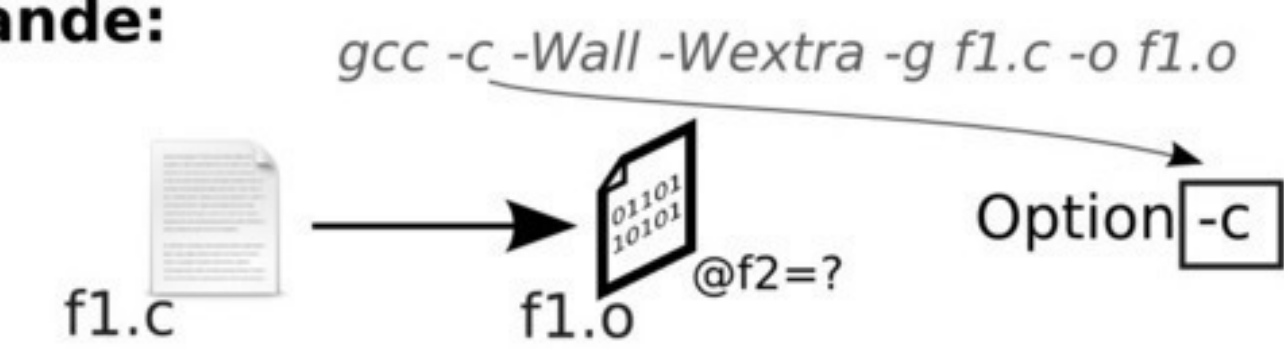


020

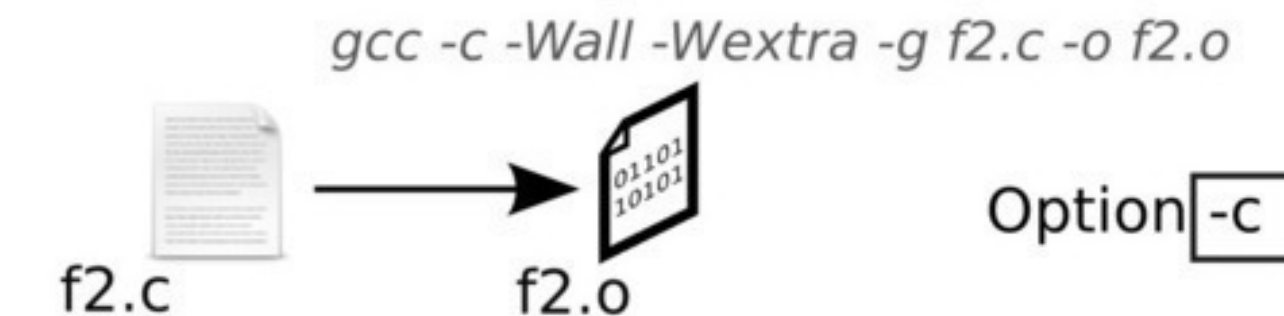
Cas de 2 fichiers: compilation séparée

Cas 2, ligne de commande:

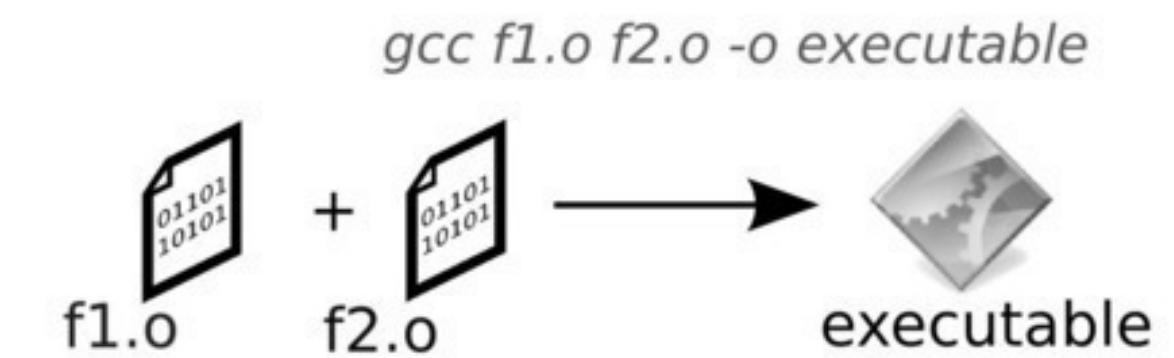
- Compilation de f1.c



- Compilation de f2.c

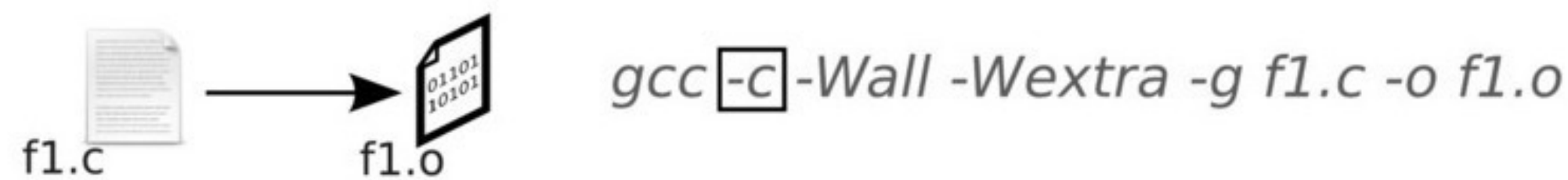


- Edition de liens, génération d'executable

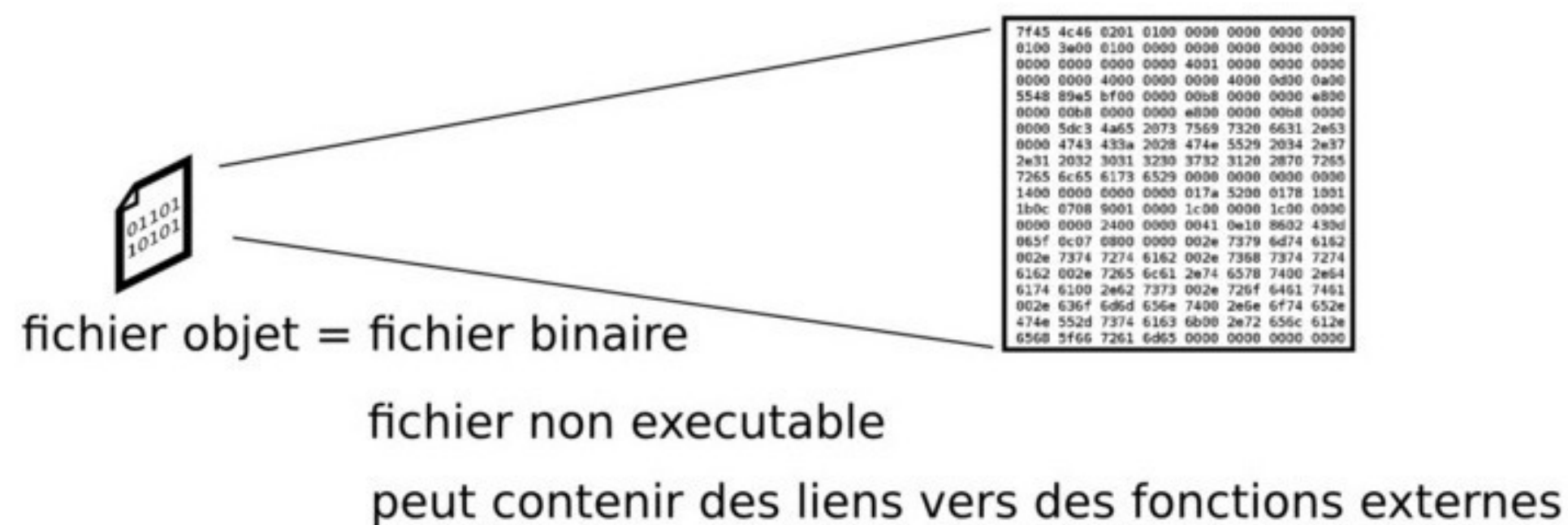


021

Fichier objet



Option **-c** indique:
génère un binaire du code
ne génère pas d'executable
s'arrête avant l'étape d'édition de lien



022

Fichier objet

Pour visualiser un fichier binaire

ex.

Ouvrir le fichier avec xemacs ou emacs
\$ xemacs fichier.o

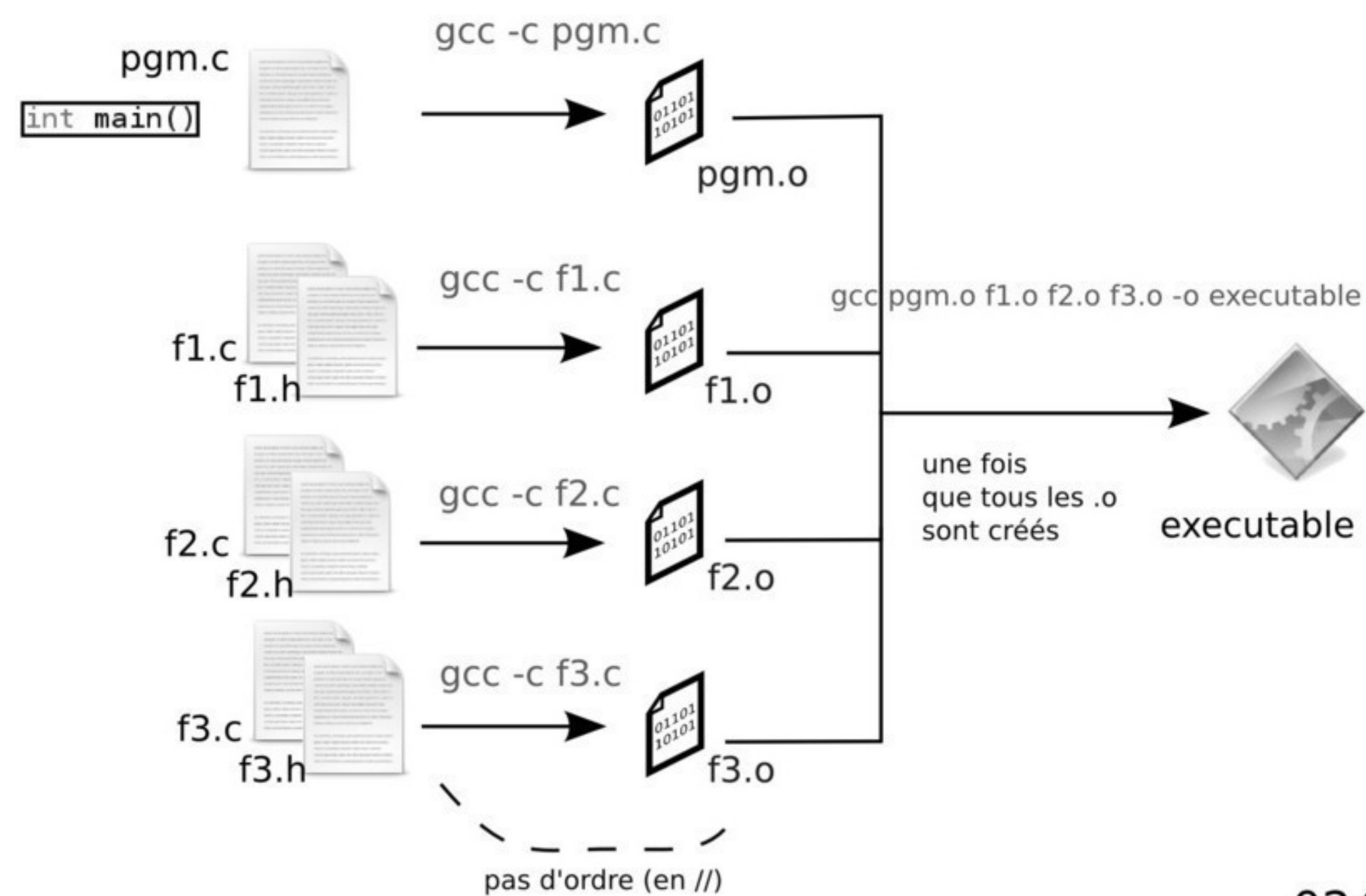
Appuyez sur **Alt+X**: cherchez *hexl-mode*

ou

\$ od -x fichier.o

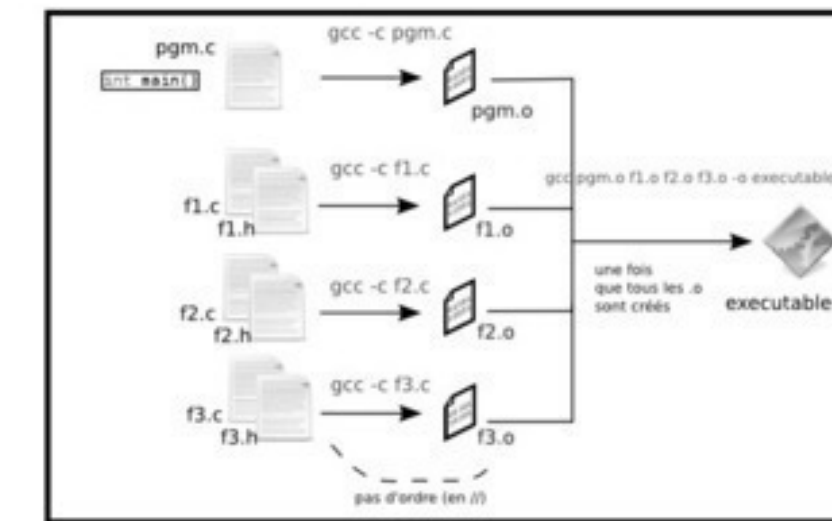
023

Plusieurs sources, un executable



024

Plusieurs sources, un executable: script



mon_script_de_compilation.sh

```
#!/bin/bash

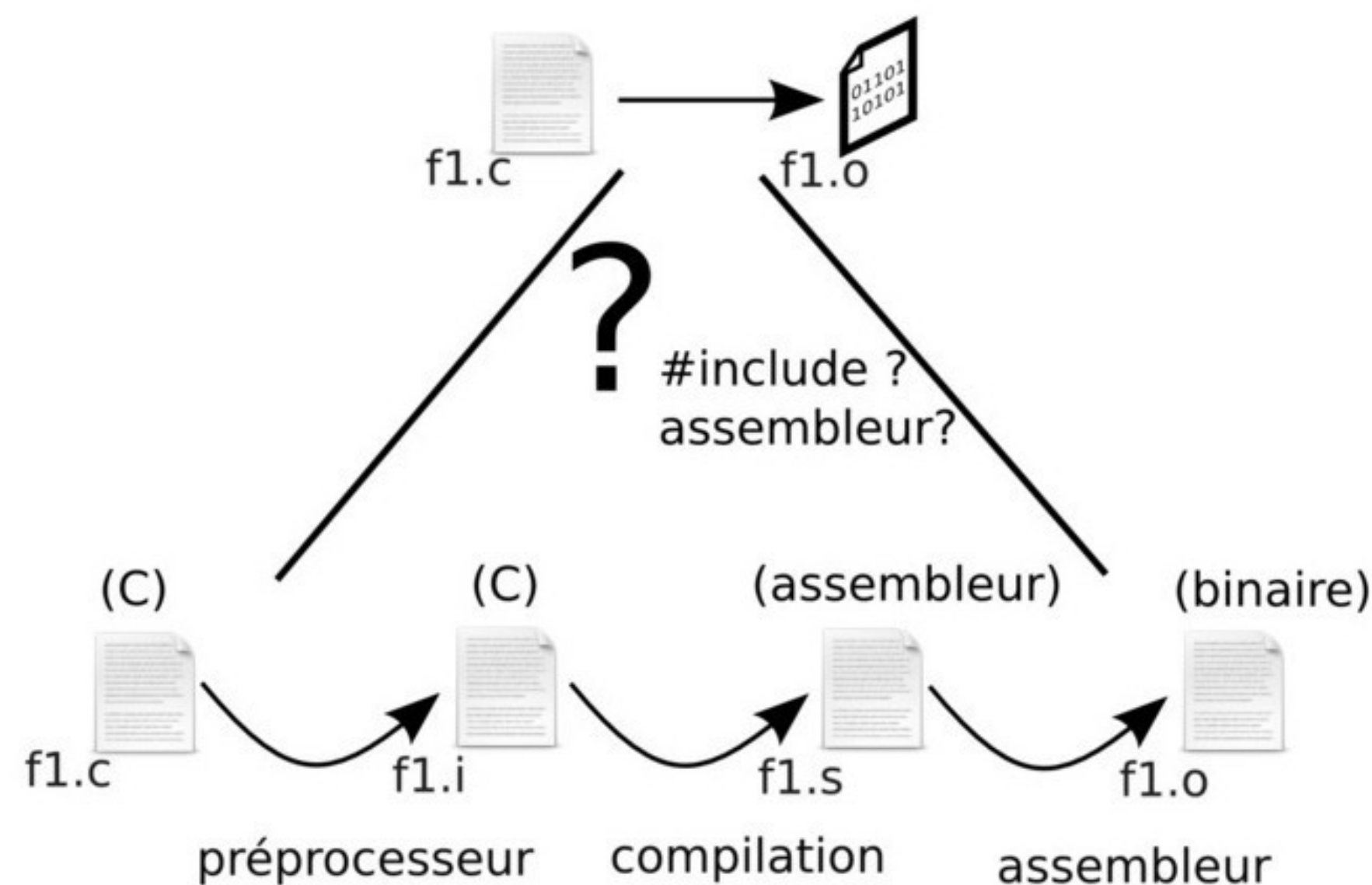
CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings -Wpointer-arith -Wcast-qual -Wredundant-decls -Winit-self -g"

#Compilation vers fichiers objets
gcc f1.c -o f1.o ${CFLAGS}
gcc f2.c -o f2.o ${CFLAGS}
gcc f3.c -o f3.o ${CFLAGS}
gcc pgm.c -o pgm.o ${CFLAGS}

#Edition de liens
gcc f1.o f2.o f3.o pgm.o -o executable
```

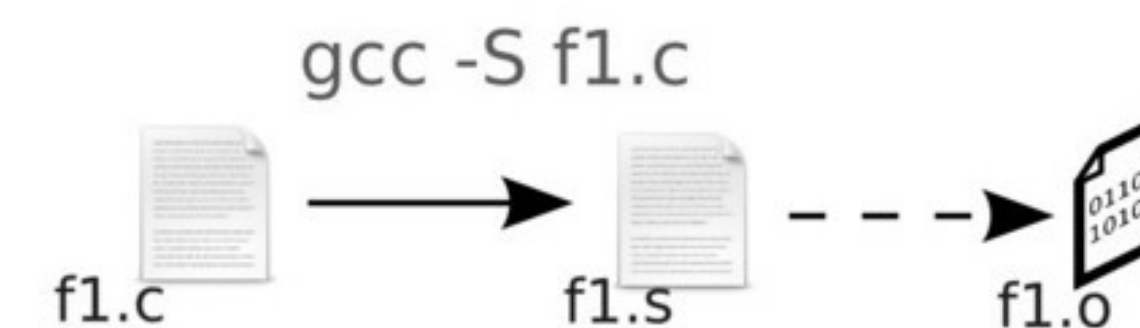
025

Construction d'un fichier objet



026

Fichier assembleur



Option **-S**

```
int main()
{
    int a=5;
    int b=6;
    int c=a+b;
}
```

```
.file "compilation.c"
.text
.globl main
.type main, @function

main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $5, -4(%rbp)
movl $6, -8(%rbp)
movl -8(%rbp), %eax
movl -4(%rbp), %edx
addl %edx, %eax
movl %eax, -12(%rbp)
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc

.LFE0:
.size main, .-main
.ident "GCC: (GNU) 4.7.1 20120721 (prerelease)"
.section .note.GNU-stack,"",@progbits
```

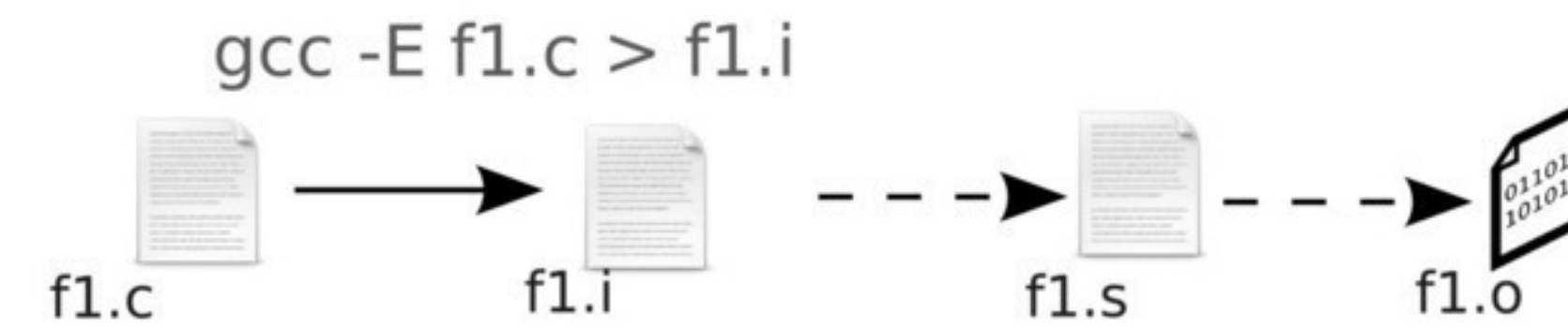
027

Chaine de compilation

Compilateur
Warnings
Compilation séparée
→ **Préprocesseur**
Makefile
Edition de liens et librairies

028

Préprocesseur

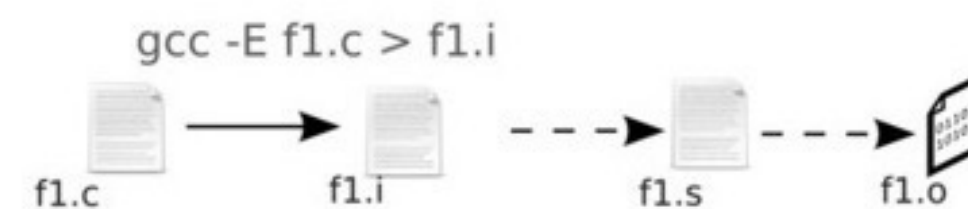


Option **-E**

Préprocesseur =
Conversion du code C en un autre code C (plus simple à compiler)

029

Préprocesseur



Elimine tous les commentaires
=> inutile pour générer de l'assembleur

f1.c

```
/**
 * Ceci est un commentaire qui devrait
 *   décrire mon programme.
 * J'aime l'informatique.
 * J'aime le C.
 */
int main()
{
    // ceci est un commentaire
    int a=5;
    /** Ceci est un autre commentaire */
    int b=6;
    int c=a+b;
}
```

gcc -E f1.c > f1.i

f1.i

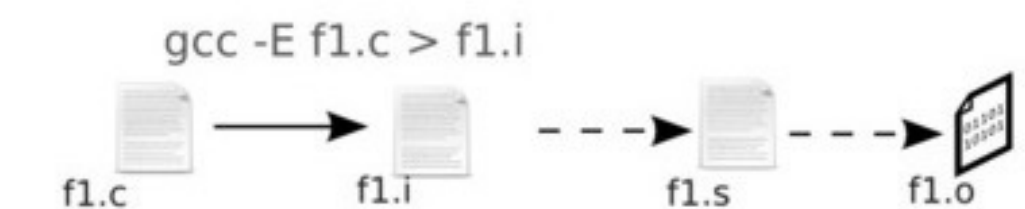
```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

int main()
{
    int a=5;
    int b=6;
    int c=a+b;
}
```

Conclusion:
Ecrivez vos commentaires pour les humains, pas pour la machine!

030

Préprocesseur



Remplace toute les *Macros*
#define

f1.c

```
#define MA_CONSTANTE 8
#define PI 3.14159

int main()
{
    int a=MA_CONSTANTE;
    int b=MA_CONSTANTE+8;
    int tableau[MA_CONSTANTE];
    float rayon=3.0;
    float aire=2*PI*RAYON;
}
```

IMPORTANT

gcc -E f1.c > f1.i

f1.i

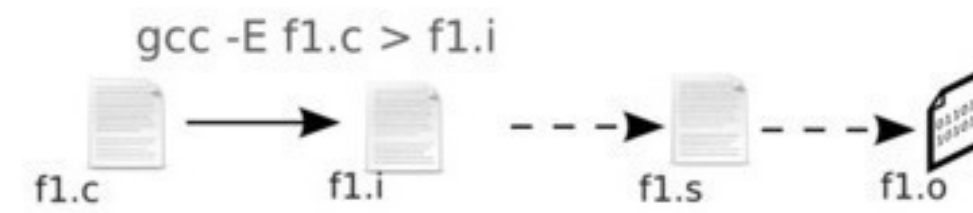
```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

int main()
{
    int a=8;
    int b=8 +8;
    int tableau[8];
    float rayon=3.0;
    float aire=2*3.14159*RAYON;
}
```

Conclusion:
Centralisez vos constantes (tailles tableaux) dans des macros.

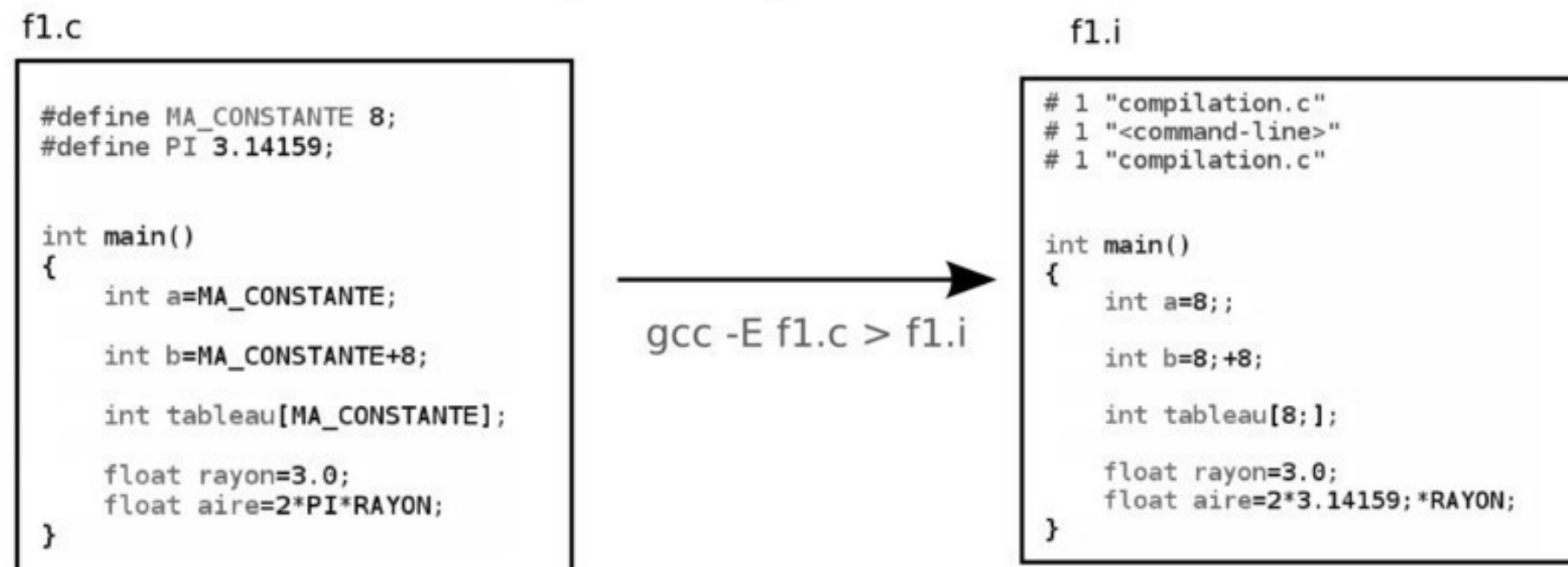
031

Preprocesseur



Remplace toute les *Macros*
#define

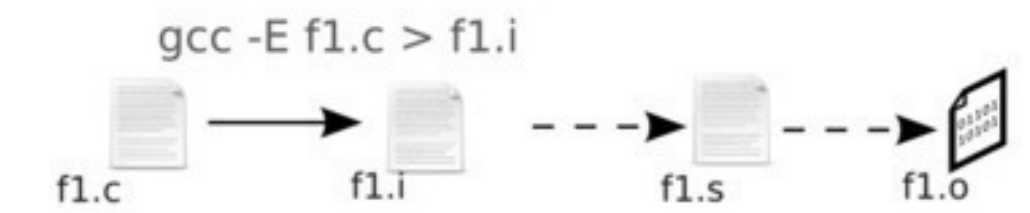
Attention: Les macros sont directement copiés collés!
=> Pas de points-virgules (;) !



Conclusion:
Pas de points-virgules dans les macros!.

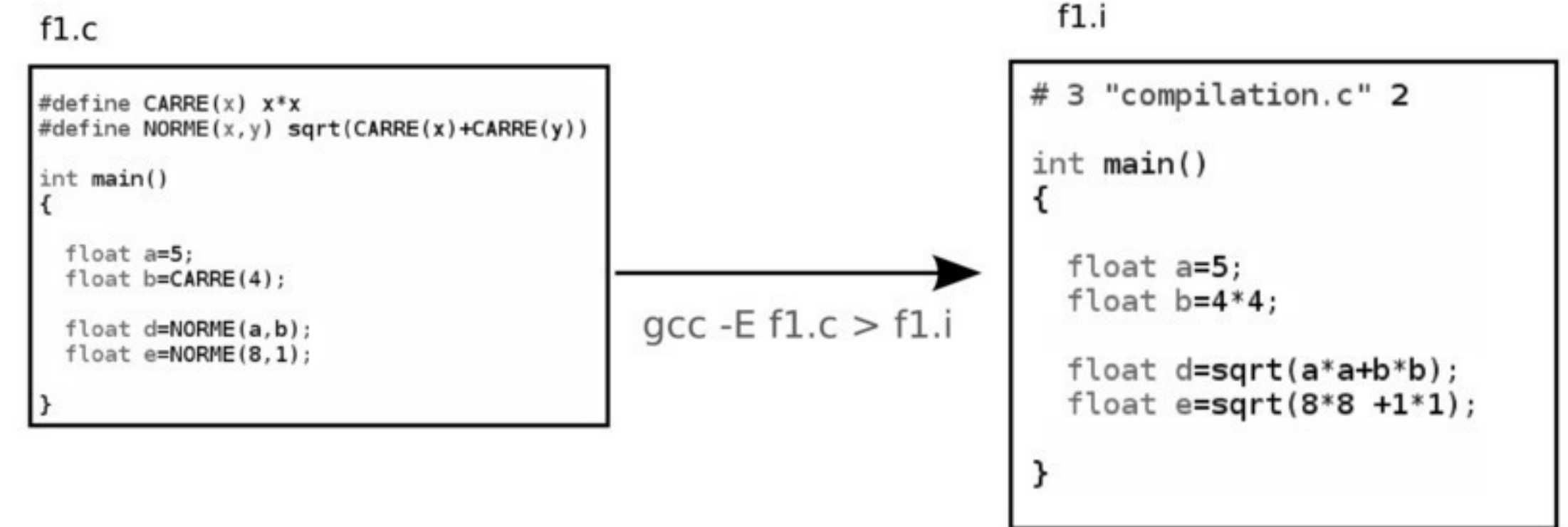
032

Preprocesseur



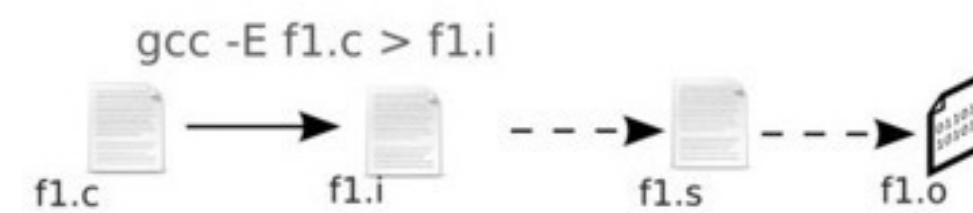
Remplace toute les *Macros*
#define

Les macros peuvent être des fonctions!



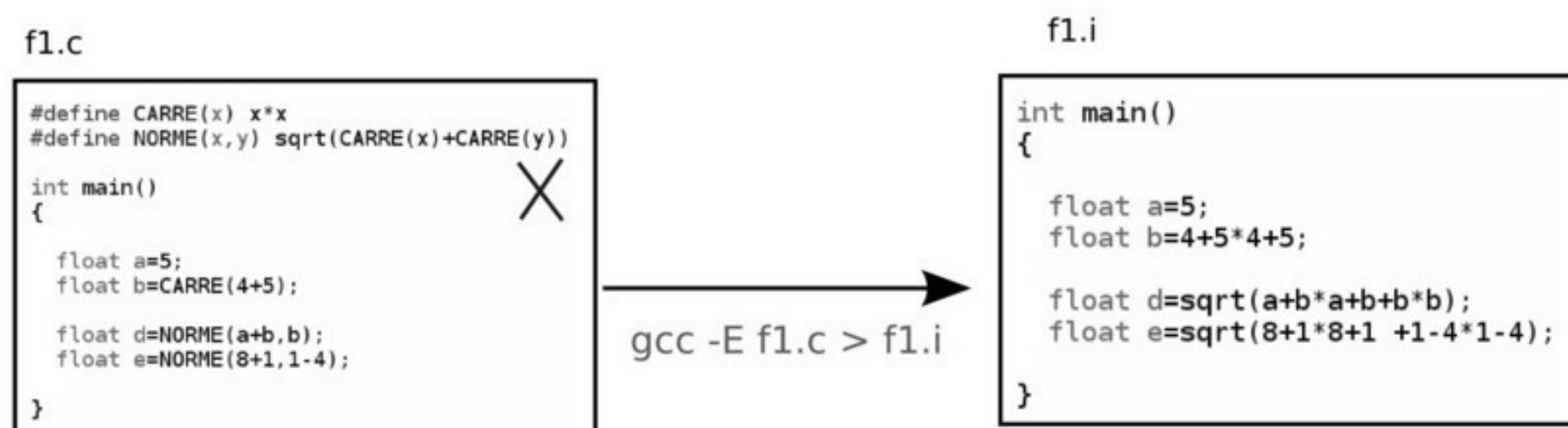
033

Preprocesseur



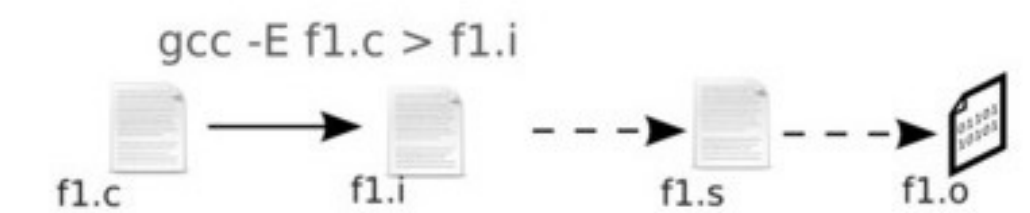
Remplace toute les *Macros*
#define

Les macros peuvent être des fonctions!
Attention au principe du copié-collé des macros !!!



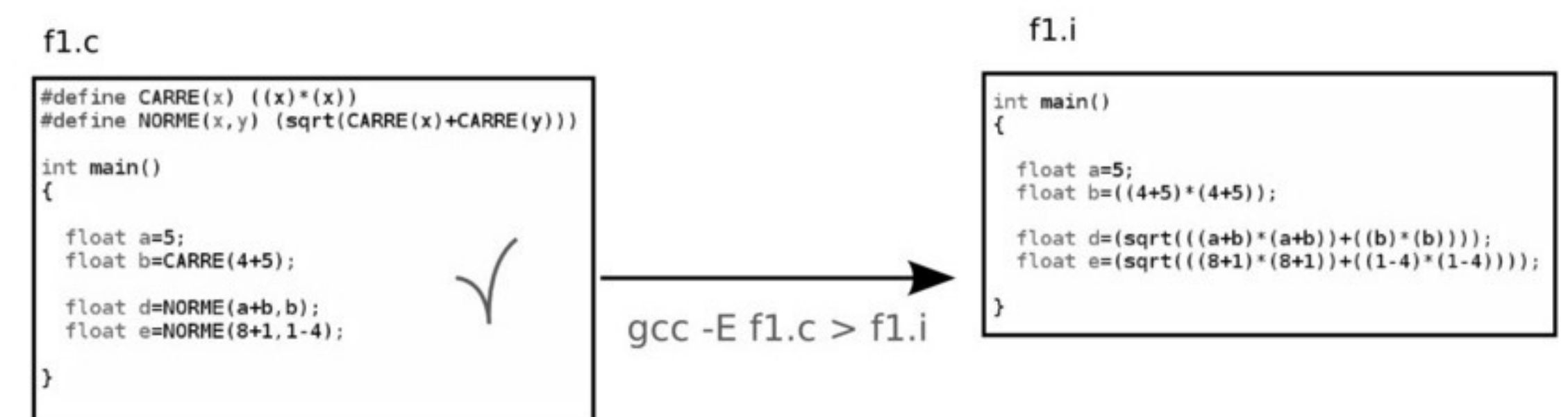
034

Preprocesseur



Remplace toutes les *Macros*
#define

Les macros peuvent être des fonctions!
Attention au principe du copié-collé des macros !!!



=> Autour de chaque variable/expression: ajoutez des parenthèses

035

Preprocesseur



gcc -E f1.c > f1.i



Remplace toutes les *Macros*
#define

Les macros peuvent être des fonctions avancées !

f1.c

```
#define LOOP(k,N) for((k)=0; (k)<(N); ++(k))
#define MAKE_VECTOR(vec,x,y,z) float vec[3];\
    vec[0]=(x);vec[1]=(y);vec[2]=(z);
#define PRINT_SUM(vec) {printf("%f\n", (vec[0])+(vec[1])+(vec[2]));}

int main()
{
    int k=0;
    LOOP(k,15)
    {
        MAKE_VECTOR(mon_vecteur,k,2*k,k)
        PRINT_SUM(mon_vecteur)
    }
}
```

gcc -E f1.c > f1.i

f1.i

```
int main()
{
    int k=0;
    for((k)=0; (k)<(15); ++(k))
    {
        float mon_vecteur[3]; mon_vecteur[0]=(k);mon_vecteur[1]=(2*k);mon_vecteur[2]=(k);
        {printf("%f\n", (mon_vecteur[0])+(mon_vecteur[1])+(mon_vecteur[2]));}
    }
}
```

036

Preprocesseur



gcc -E f1.c > f1.i



Nouveau langage?

f1.c

```
DEBUT_PROGRAMME

Initialise(Valeur_utilisateur)
Variable A Reçoit Valeur_utilisateur;

SI(A Plus_grand_que 10)
    Affiche("Trop grand");
    Quitte
FIN_SI

Variable K Reçoit 0;
TANT_QUE(K Plus_petit_que A)
    Affiche(K)
    Incrmente(K);
FIN_TANT_QUE

FIN_PROGRAMME
```

```
+
#define DEBUT_PROGRAMME int main(){
#define FIN_PROGRAMME return 0;}
#define Reçoit =
#define Variable int
#define SI(x) if(x){
#define Plus_grand_que >
#define Plus_petit_que <
#define Affiche(x) {printf("%d\n", (x));}
#define Incrmente(x) x=x+1
#define TANT_QUE(x) while(x){
#define FIN_SI }
#define FIN_TANT_QUE }
#define Initialise(x) int x;\
    scanf("%d",&x);
#define Quitte abort();
```

gcc -E f1.c > f1.i

f1.i

```
int main(){
    int Valeur_utilisateur; scanf("%d",&Valeur_utilisateur);
    int A = Valeur_utilisateur;

    if(A > 10){
        {printf("%d\n", ("Trop grand"));}
        abort();
    }

    int K = 0;
    while(K < A){
        {printf("%d\n", (K));}
        K=K+1;
    }

    return 0;}
}
```

037

Preprocesseur

Synthèse Macros:



gcc -E f1.c > f1.i



Avantage: Pas de types

Inconvénients: Difficile à écrire
Difficile à debugger
Cas non prévues

Macro = Attention!

Ne pas abuser des macros

Règles: Ne pas utiliser une macro si une fonction peut le faire!

Ne pas utiliser de macro si il y a ambiguïté

Ne pas utiliser de macro si le code est moins lisible

038

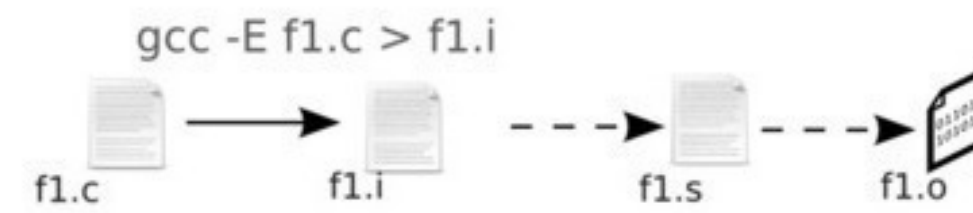
Preprocesseur

Quelques macros utiles

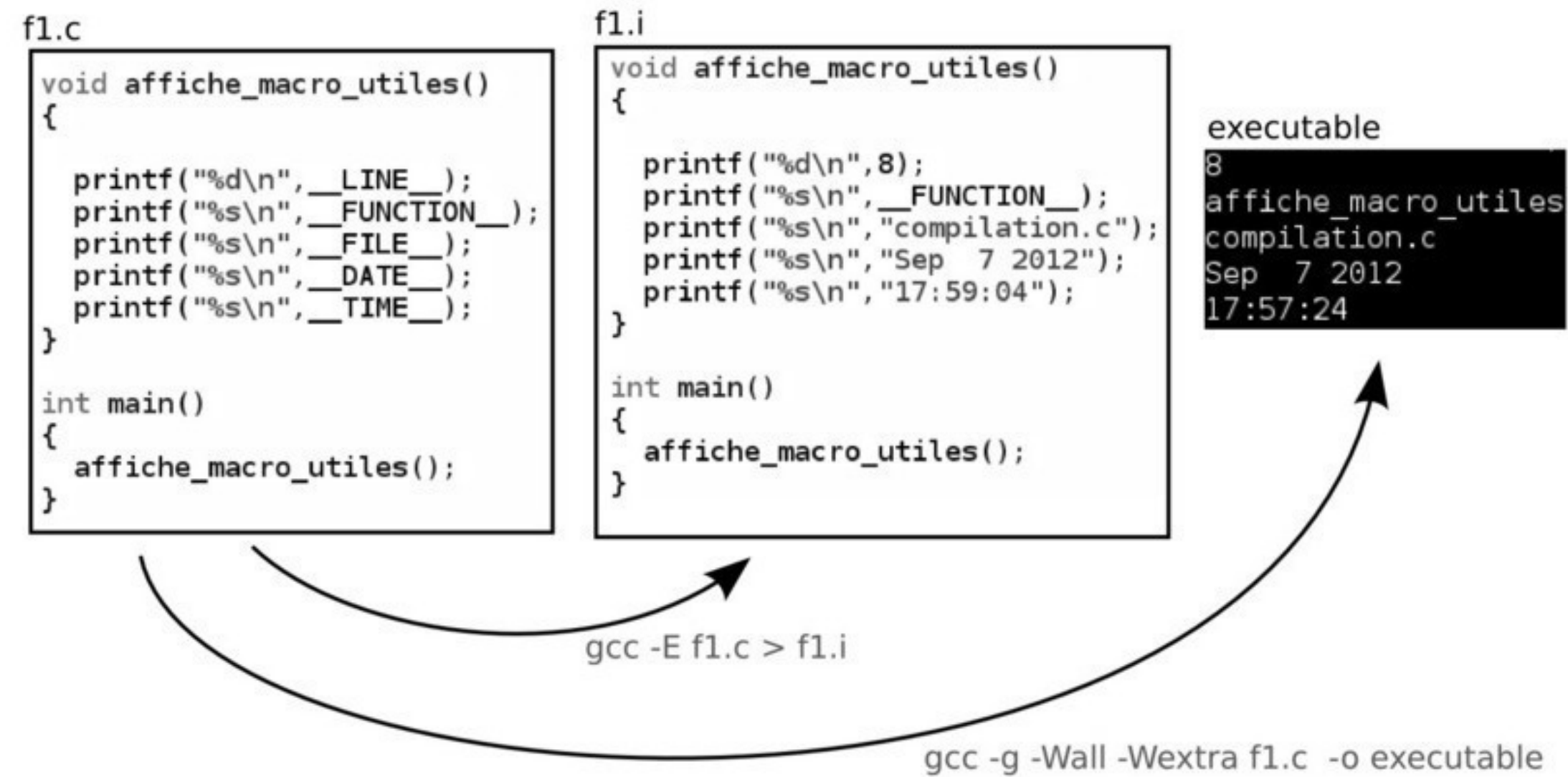
```
__LINE__,
__FUNCTION__,
__FILE__,
__DATE__
```

039

Preprocesseur



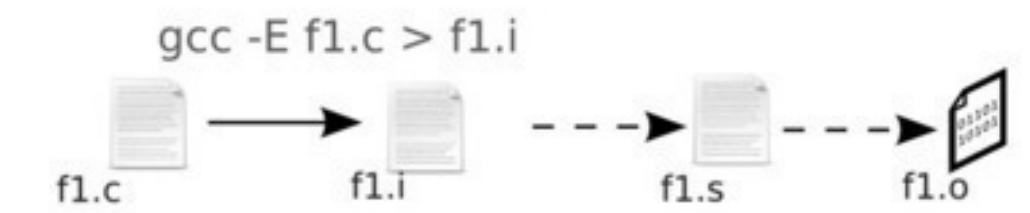
Quelques macros utiles: exemple d'utilisation



Rem. `__FUNCTION__` n'est pas une vraie macro, mais une variable liée à gcc

040

Preprocesseur



Quelques macros utiles: exemple d'utilisation

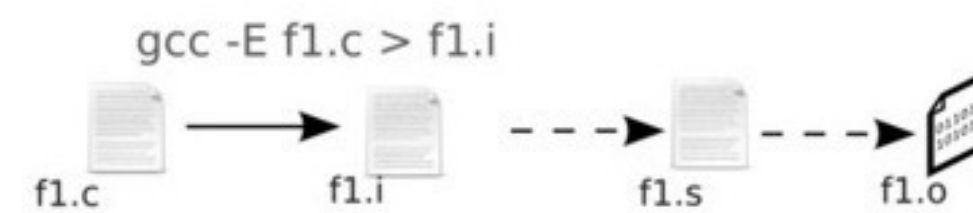
Utile pour messages d'erreurs!

```

void ma_fonction(int T[])
{
    //boucle complexe
    int k=0;
    for(k=0; ...)
    {
        ...
        //detection d'erreur
        if(erreur)
        {
            printf("Erreur detectee ligne %d, fonction %s, fichier %s\n",
                __LINE__, __FUNCTION__, __FILE__);
        }
        ...
    }
}
  
```

041

Preprocesseur



Instructions contionnelles du preprocesseur

```

#if <condition>
<...>
#endif
  
```

//si condition est vraie

```

#if <condition>
<...>
#else
<...>
#endif
  
```

//si condition est vraie
//sinon

```

#if <condition>
<...>
#elif <conditon 2>
<...>
#else
<...>
#endif
  
```

//si condition est vraie
//sinon, si condition 2 est vraie
//sinon

```

#ifdef <variable>
<...>
#endif
  
```

//si variable definie

```

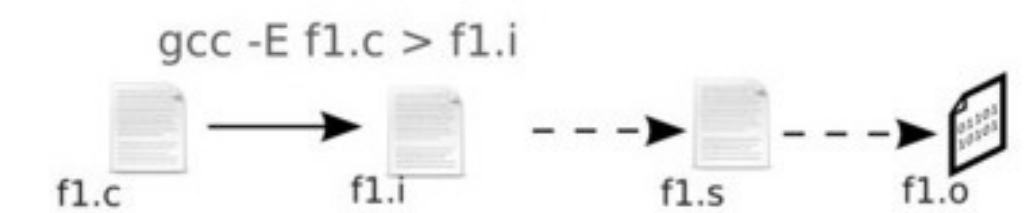
#ifndef <variable>
<...>
#endif
  
```

//si variable non definie

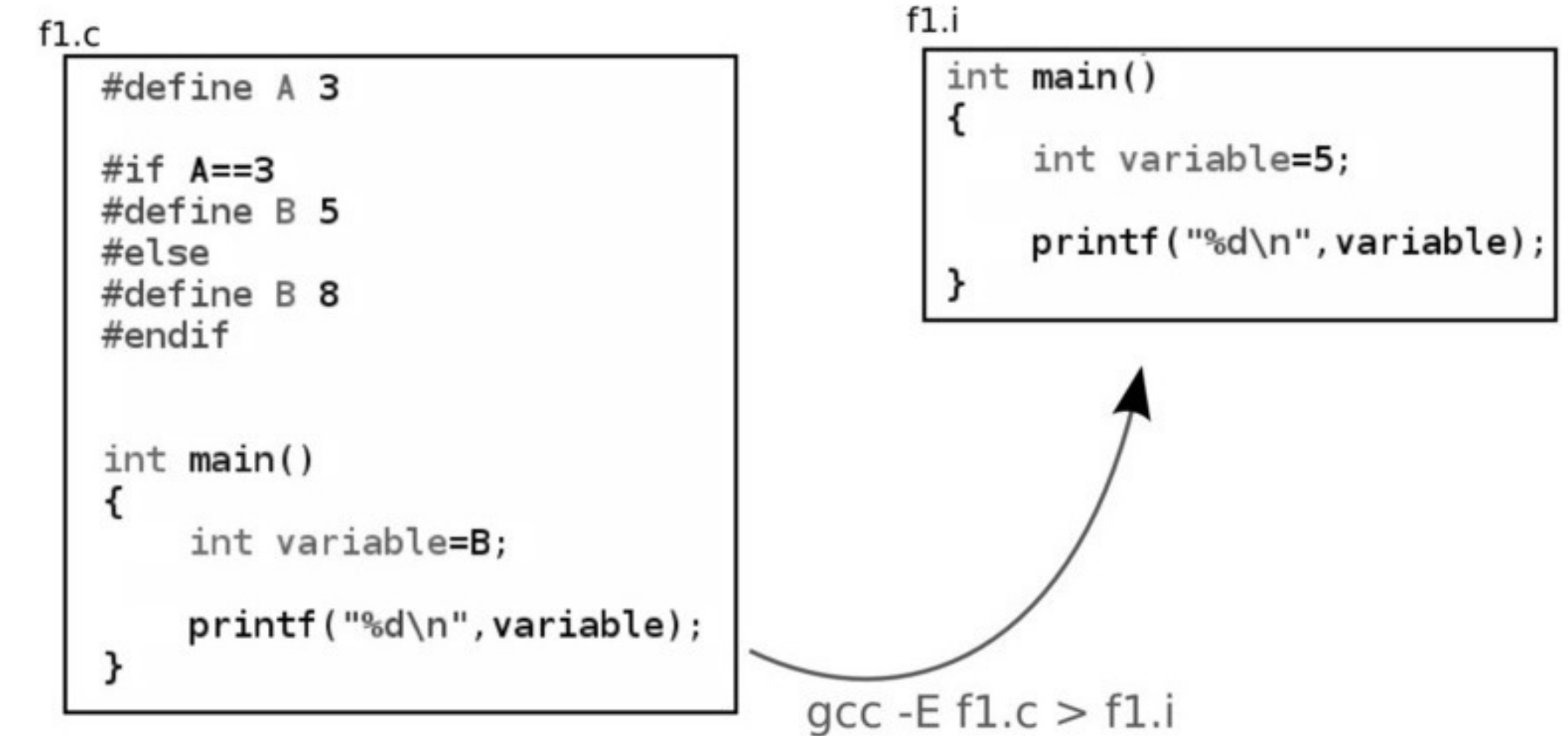


042

Preprocesseur



Exemple:



043

Preprocesseur



Exemple:
f1.c

```
#define AVEC_MATH
#ifdef AVEC_MATH
#include <math.h>
#endif
int main()
{
    float x=1e-5;
#ifdef AVEC_MATH
    float sin_x=x;
#else
    float sin_x=sin(x);
#endif
}
```

f2.c

```
#ifdef AVEC_MATH
#include <math.h>
#endif
int main()
{
    float x=1e-5;
#ifdef AVEC_MATH
    float sin_x=x;
#else
    float sin_x=sin(x);
#endif
}
```

gcc -E f1.c > f1.i

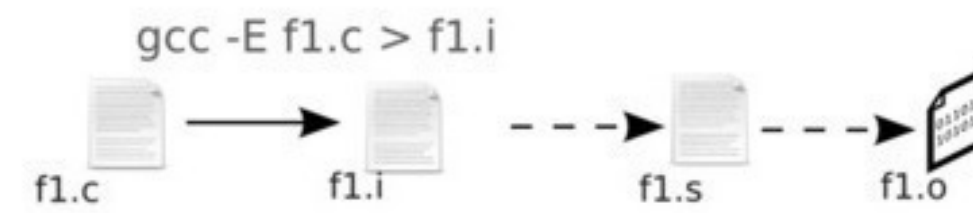
gcc -E f2.c > f2.i

f1.i

```
int main()
{
    float x=1e-5;
    float sin_x=sin(x);
}
```

f2.i

```
int main()
{
    float x=1e-5;
    float sin_x=x;
}
```



044

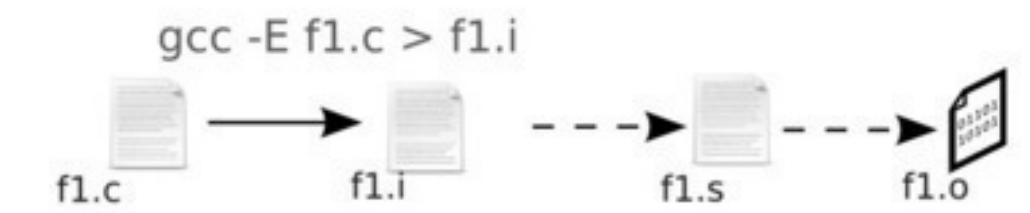
Preprocesseur

Exemple:

```
#define DEBUG_MODE
#ifdef DEBUG_MODE
#define PRINT_DEBUG printf("Erreur: l.%d, fct:<%s>, fichier:[%s] \n", __LINE__, __FUNCTION__, __FILE__)
#else
#define PRINT_DEBUG
#endif
int divide(int a,int b)
{
    if(b!=0)
        return a/b;
    else
    {
        PRINT_DEBUG;
        return -1;
    }
}
int main()
{
    divide(5,0);
    return 0;
}
```

```
int divide(int a,int b)
{
    if(b!=0)
        return a/b;
    else
    {
        printf("Erreur: l.%d, fct:<%s>, fichier:[%s] \n",17,__FUNCTION__,"compilation.c");
        return -1;
    }
}
int main()
{
    divide(5,0);
    return 0;
}
```

gcc -E f1.c > f1.i



045

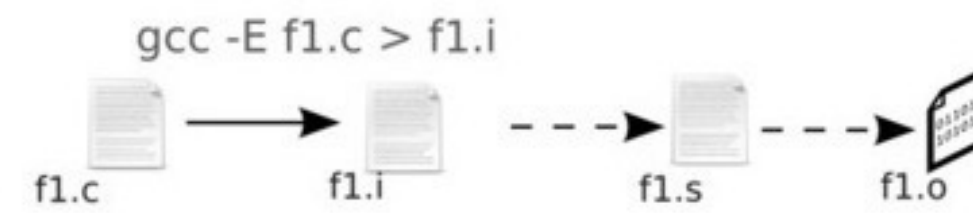
Preprocesseur

Utilisation des conditionnelles+macro:

Pour la portabilité entre systèmes

Pour le debug

```
#if SYSTEM==LINUX
#include <stdio.h>
#else if SYSTEM==WINDOWS
#include <stdafx.h>
#endif
#if ARCHI==x86
...
#else if ARCHI==itanium
...
#else if ARCHI==power_pc
...
#endif
#if NBITS==64
...
#else if NBITS==32
...
#endif
```



Attention: Ne **pas** utiliser pour l'optimisation

Réduit la lisibilité du code.

Si faisable avec une fonction: **utiliser une fonction**

046

Preprocesseur

Macro: #include "fichier"

Copie-colle le contenu d'un fichier



```
mon_fichier
void ma_fonction(int a)
{
    printf("bonjour %d\n",a);
}
int ma_variable=36;
```

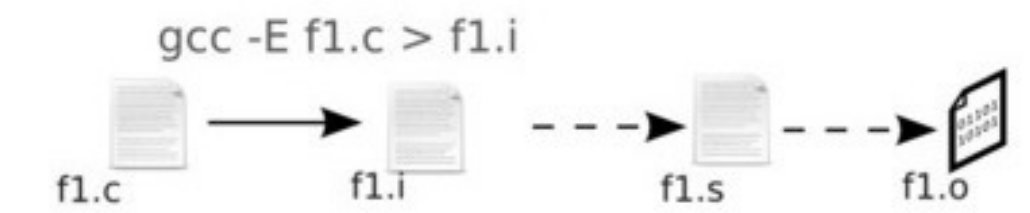
```
f1.c
#include "mon_fichier"
int main()
{
    ma_fonction(ma_variable);
    return 0;
}
```

```
f1.i
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"
# 1 "mon_fichier" 1
void ma_fonction(int a)
{
    printf("bonjour %d\n",a);
}
int ma_variable=36;
# 3 "compilation.c" 2
int main()
{
    ma_fonction(ma_variable);
    return 0;
}
```

execution:

bonjour 36

gcc -E f1.c > f1.i



047

Preprocesseur

gcc -E f1.c > f1.i



En-tête standards

f1.c

```
#include <stdio.h>

int main()
{
}
```

f1.i

```
...
extern int fprintf (FILE *__restrict __stream,
  const char *__restrict __format, ...);

extern int printf (const char *__restrict __format, ...);

extern int sprintf (char *__restrict __s,
  const char *__restrict __format, ...) __attribute__ ((__nothrow__));
...
```

gcc -E f1.c > f1.i



048

Preprocesseur

gcc -E f1.c > f1.i



En-tête standards

f1.c

```
#include <stdio.h>

int main()
{
}
```

f1.i

```
...
extern int fprintf (FILE *__restrict __stream,
  const char *__restrict __format, ...);

extern int printf (const char *__restrict __format, ...);

extern int sprintf (char *__restrict __s,
  const char *__restrict __format, ...) __attribute__ ((__nothrow__));
...
```

gcc -E f1.c > f1.i

Note: #include "NOM" recherche fichier dans le repertoire locale

#include <NOM> recherche fichier dans les repertoires systemes (/usr/include/ ; /usr/local/include/ ; ...)



Pour inclure d'autres chemins:
gcc -I<CHEMIN>

049

Preprocesseur

gcc -E f1.c > f1.i



En-tête standards

f1.c

```
#include <stdio.h>

int main()
{
}
```

f1.i

```
...
extern int fprintf (FILE *__restrict __stream,
  const char *__restrict __format, ...);

extern int printf (const char *__restrict __format, ...);

extern int sprintf (char *__restrict __s,
  const char *__restrict __format, ...) __attribute__ ((__nothrow__));
...
```

gcc -E f1.c > f1.i

Note: Les fichiers d'en tête standards sont volumineux
=> N'inclure que ceux qui sont nécessaire

Les variables des fichiers standards commencent par _ ou __
=> Ne pas utiliser cette convention pour vos variables

050

Preprocesseur

gcc -E f1.c > f1.i



Inclusion multiples: include guard

voiture.h

```
//voiture de location
struct voiture
{
  int essence;
  int kilometrage;
};

void voiture_init(struct voiture* v);
```

voiture.c

```
#include "voiture.h"

void voiture_init(struct voiture* v)
{
  v->kilometrage=0;
  v->essence=60;
}
```

trajet.h

```
#include "voiture.h"

void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);
```

trajet.c

```
#include "trajet.h"

void trajet_lyon_paris(struct voiture* v)
{
  v->essence -= 30;
  v->kilometrage += 400;
}

void trajet_marseille_lyon(struct voiture* v)
{
  v->essence -= 21;
  v->kilometrage += 300;
}
```

main.c

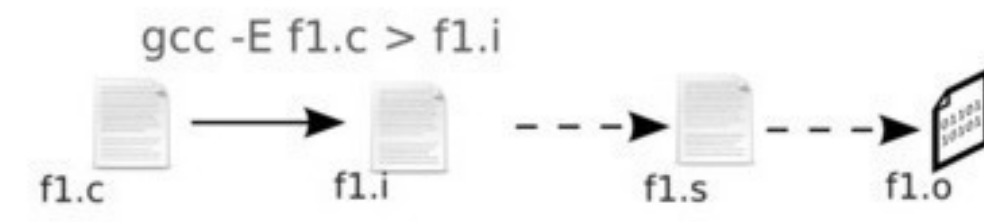
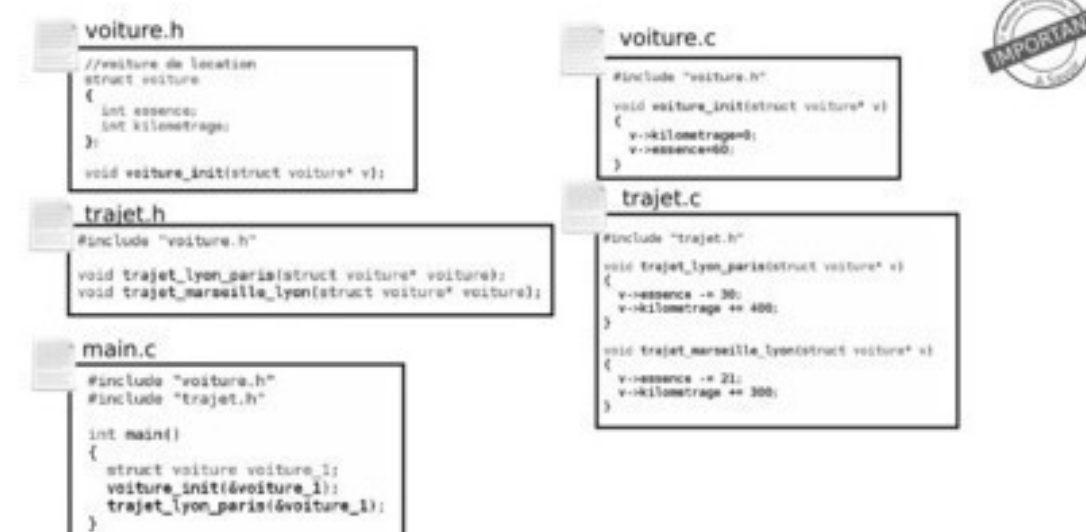
```
#include "voiture.h"
#include "trajet.h"

int main()
{
  struct voiture voiture_1;
  voiture_init(&voiture_1);
  trajet_lyon_paris(&voiture_1);
}
```

051

Preprocesseur

Inclusion multiples: *include guard*

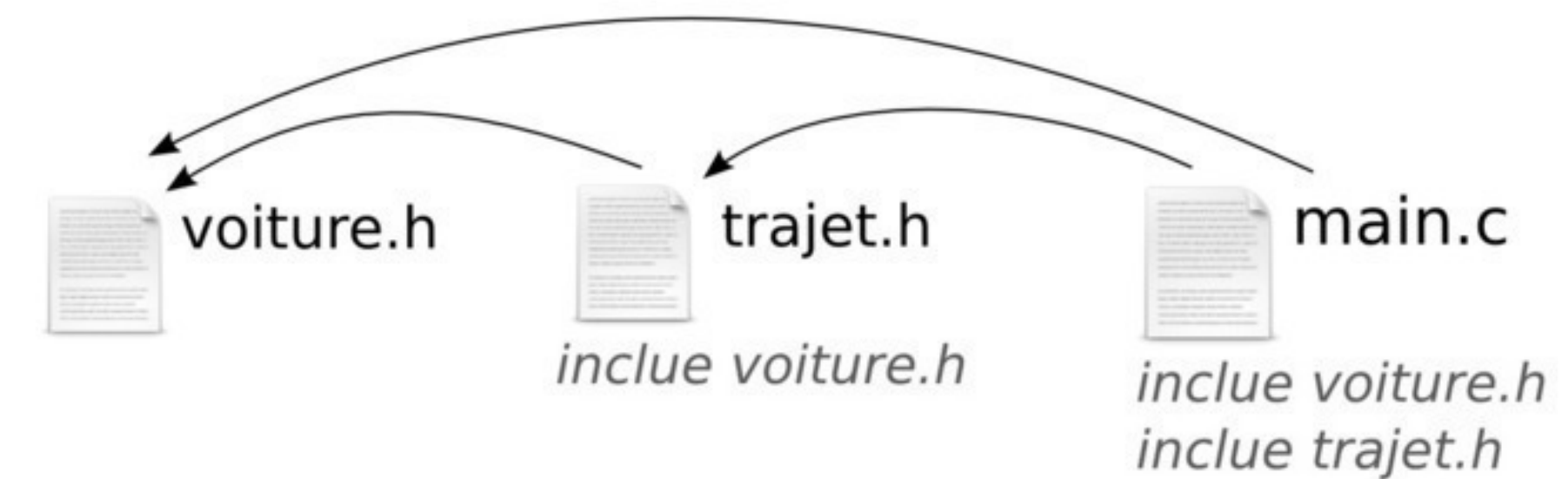
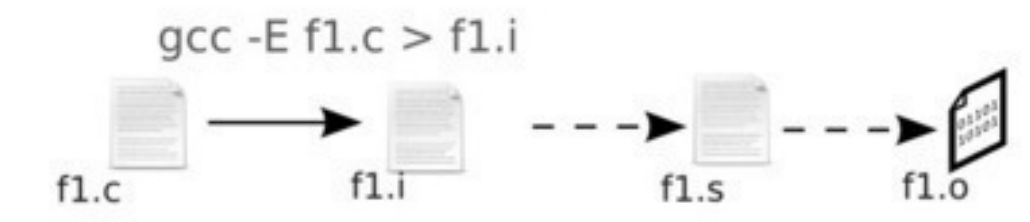
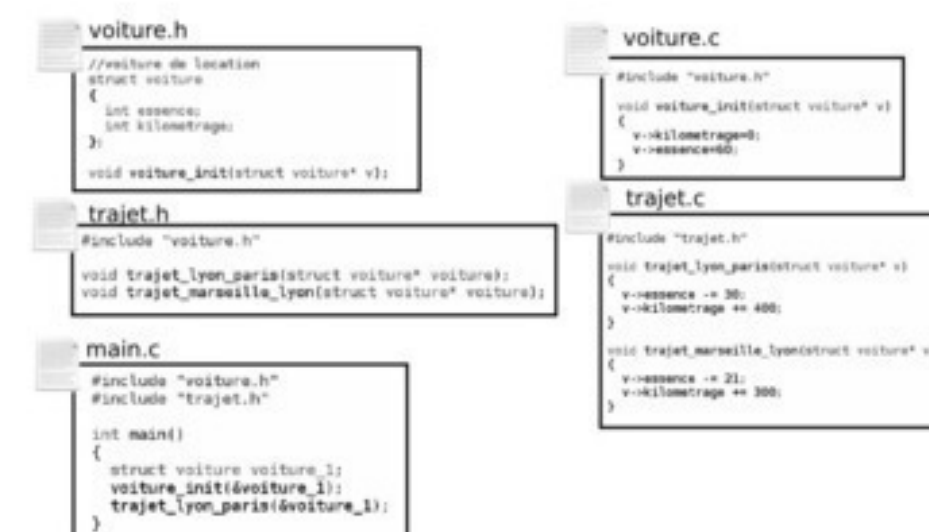


```
main.i
# 1 "f1.c"
# 1 "<command-line>"
# 1 "f1.c"
# 1 "voiture.h" 1
# 1 "trajet.h" 1
# 1 "voiture.h" 1
struct voiture
{
    int essence;
    int kilometrage;
};
void voiture_init(struct voiture* v);
# 3 "f1.c" 2
# 1 "trajet.h" 2
# 1 "voiture.h" 2
struct voiture
{
    int essence;
    int kilometrage;
};
void voiture_init(struct voiture* v);
# 6 "trajet.h" 2
void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);
# 4 "f1.c" 2
int main()
{
    struct voiture voiture_1;
    voiture_init(&voiture_1);
    trajet_lyon_paris(&voiture_1);
}
```

052

Preprocesseur

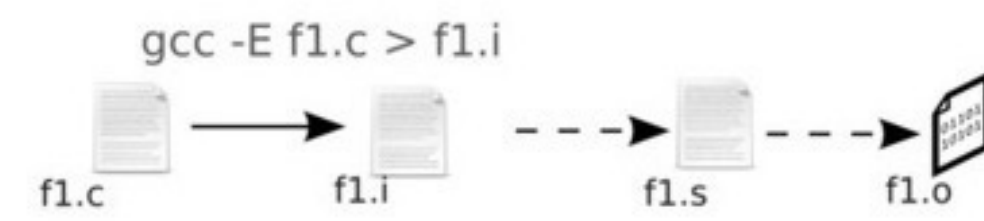
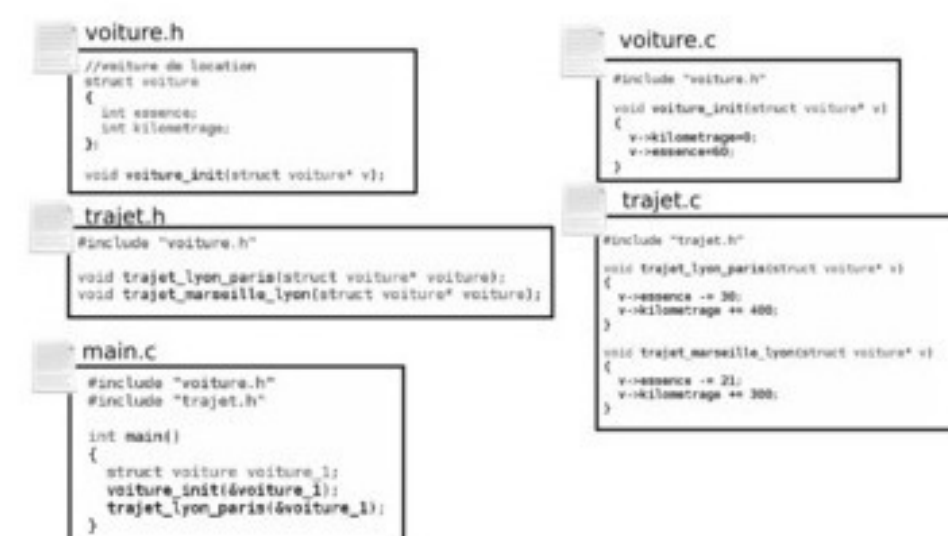
Inclusion multiples: *include guard*



053

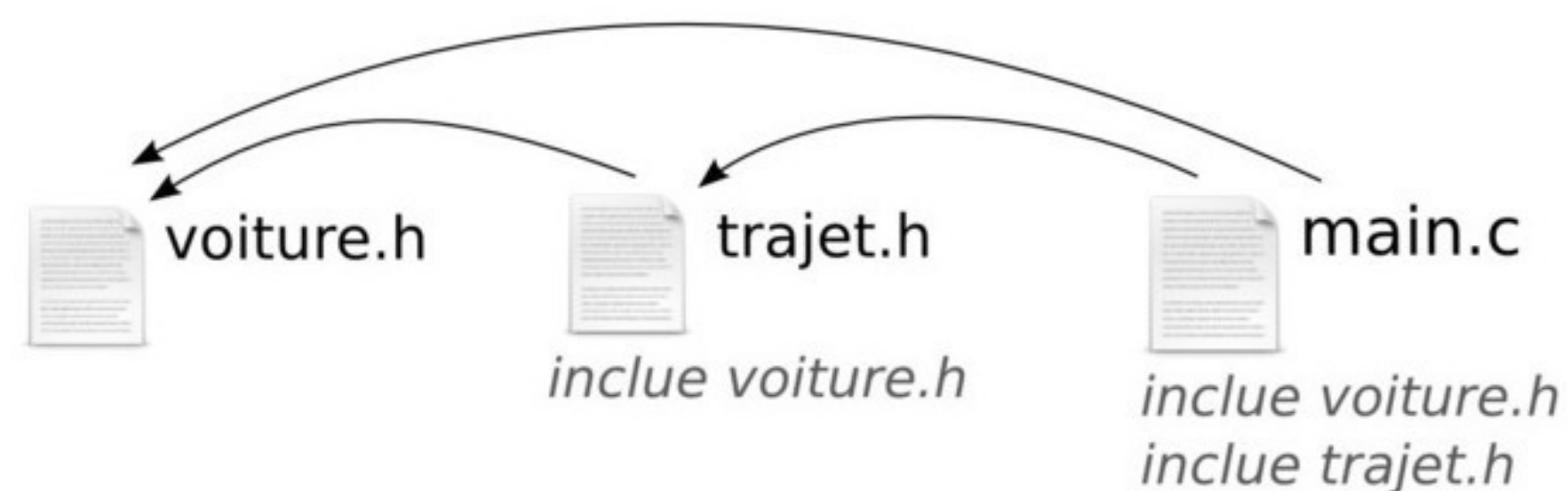
Preprocesseur

Inclusion multiples: *include guard*



Solution 1:
N'inclure que trajet.h dans main.c
=> Ingérable pour un grand projet

Solution 2:
Compilateur n'inclue pas voiture.h deux fois
=> Principe des *includes guards*



054

Preprocesseur

Inclusion multiples: *include guard*

```
voiture.h
#ifndef VOITURE_H
#define VOITURE_H

//voiture de location
struct voiture
{
    int essence;
    int kilometrage;
};

void voiture_init(struct voiture* v);

#endif
```

une unique inclusion



```
trajet.h
#include "voiture.h"
void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);

main.c
#include "voiture.h"
#include "trajet.h"
int main()
{
    struct voiture voiture_1;
    voiture_init(&voiture_1);
    trajet_lyon_paris(&voiture_1);
}
```

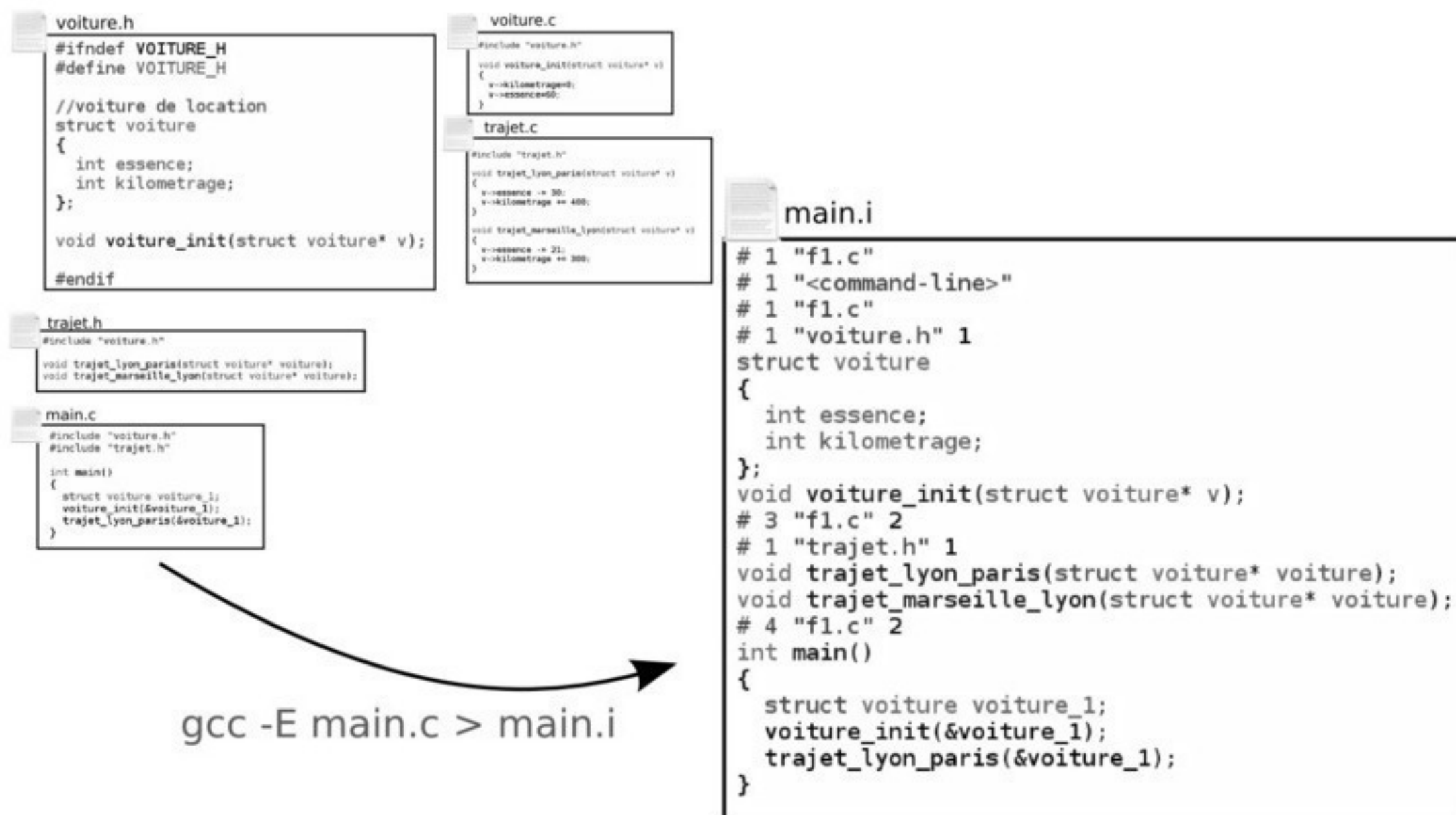
```
voiture.c
#include "voiture.h"
void voiture_init(struct voiture* v)
{
    v->essence = 30;
    v->kilometrage = 400;
}

trajet.c
#include "trajet.h"
void trajet_lyon_paris(struct voiture* voiture)
{
    v->essence = 30;
    v->kilometrage = 400;
}
```

055

Preprocesseur

Inclusion multiples: *include guard*



056

Preprocesseur

Inclusion multiples: *include guard*

Bonne pratique:

Pour tout fichier d'en-tête de nom: NOM.h
Placez un *include guard*

```

#ifndef NOM_H
#define NOM_H
...
#endif
    
```

Autres possibilités:

```

#ifndef INCLUDE_GUARD_NOM_H
#ifndef NOM
    
```

But: Nom éviter les collisions

Attention: Evitez les macros commençant par _

~~#ifndef _NOM_H~~

057

Preprocesseur

Exemple réel:
fichier unistd.h

```

/*
 * POSIX Standard: 2.10 Symbolic Constants <unistd.h>
 */
#ifndef _UNISTD_H
#define _UNISTD_H 1
#include <features.h>
__BEGIN_DECLS
/* These may be used to determine what facilities are present at compile time
   Their values can be obtained at run time from 'sysconf'. */
#ifdef __USE_XOPEN2K8
/* POSIX Standard approved as ISO/IEC 9945-1 as of September 2008. */
#define _POSIX_VERSION 200809L
#elif defined __USE_XOPEN2K
/* POSIX Standard approved as ISO/IEC 9945-1 as of December 2001. */
#define _POSIX_VERSION 200112L
#elif defined __USE_POSIX199506
/* POSIX Standard approved as ISO/IEC 9945-1 as of June 1995. */
#define _POSIX_VERSION 199506L
#elif defined __USE_POSIX199309
/* POSIX Standard approved as ISO/IEC 9945-1 as of September 1993. */
#define _POSIX_VERSION 199309L
#else
/* POSIX Standard approved as ISO/IEC 9945-1 as of September 1990. */
#define _POSIX_VERSION 199009L
#endif
/* These are not #ifdef __USE_POSIX2 because they are
   in the theoretically application-owned namespace. */
#ifdef __USE_XOPEN2K8
#define _POSIX2_THIS_VERSION 200809L
/* The utilities on GNU systems also correspond to this version. */
#elif defined __USE_XOPEN2K
/* The utilities on GNU systems also correspond to this version. */
#define _POSIX2_THIS_VERSION 200112L
#elif defined __USE_POSIX199506
...
    
```

058

Preprocesseur

Synthèse préprocesseur

Potentiellement puissant (va au delà du langage C)

A utiliser avec précaution

=> rend rapidement le code peu standard
= peu lisible

Bonne pratique:

N'utilisez que les règles standards:

- * include guard
- * #ifdef => portabilité, debug.
- * pas d'optimisation, préférez les fonctions aux macros

059

Chaine de compilation

Compilateur
Warnings
Compilation séparée
Préprocesseur
→ **Makefile**
Edition de liens et librairies

060

Makefile

make: Outil d'automatisation de tâches
Dépendance => Action à réaliser

Note: Makefile est indépendant de gcc!



Makefile

```
but_1: dependance but 1
      action a realiser pour le but_1

but_2: dependance but 2
      action a realiser pour le but_2
```

061

Makefile



Exemple de Makefile

```
#par default, ce Makefile sert a realiser un fichier du nom de mon_rapport.txt
all: mon_rapport.txt

mon_rapport.txt: partie_1.txt partie_2.txt
    cp partie_1.txt mon_rapport.txt #copie de fichier
    cat partie_2.txt >> mon_rapport.txt #concatenation fichier dans mon_rapport.txt

partie_1.txt:
    echo "Chapitre I: " > partie_1.txt #ecriture dans un fichier
    echo "Je realise mon 1er Makefile" >> partie_1.txt

partie_2.txt:
    echo "Chapitre II: " > partie_2.txt
    echo "Je compile mon programme" >> partie_2.txt
```

L'appel à: \$ make
génère:



partie_1.txt



partie_2.txt



mon_rapport.txt

062

Makefile



Exemple de Makefile pour compiler du C

variable

```
CFLAGS=-g -Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings -Wpointer-arith -Wcast-qual -Wredundant-decls -Winit-self

all: mon_executable

mon_executable: f1.o f2.o f3.o
    gcc f1.o f2.o f3.o -o mon_executable

f1.o: f1.c f1.h

f2.o: f2.c f2.h

f3.o: f3.c f3.h
```

Avantage par rapport à un script:
Ne compile que si nécessaire

063

Makefile

Exemple de Makefile pour compiler du C

```
all: mon_executable
mon_executable: f1.o f2.o f3.o
    gcc f1.o f2.o f3.o -o mon_executable

#generation des fichiers assembleurs
assembleur: f1.s f2.s f3.s

f1.o: f1.c f1.h
f2.o: f2.c f2.h
f3.o: f3.c f3.h

#compilation pour assembleur
f1.s: f1.c f1.h
    gcc -c -S f1.c
f2.s: f2.c f2.h
    gcc -c -S f2.c
f3.s: f3.c f3.h
    gcc -c -S f3.c
```

fichiers assembleurs
pas de commandes d'executions

L'appel à
\$ make assembleur
génère les fichiers:
f1.s, f2.s, f3.s

064

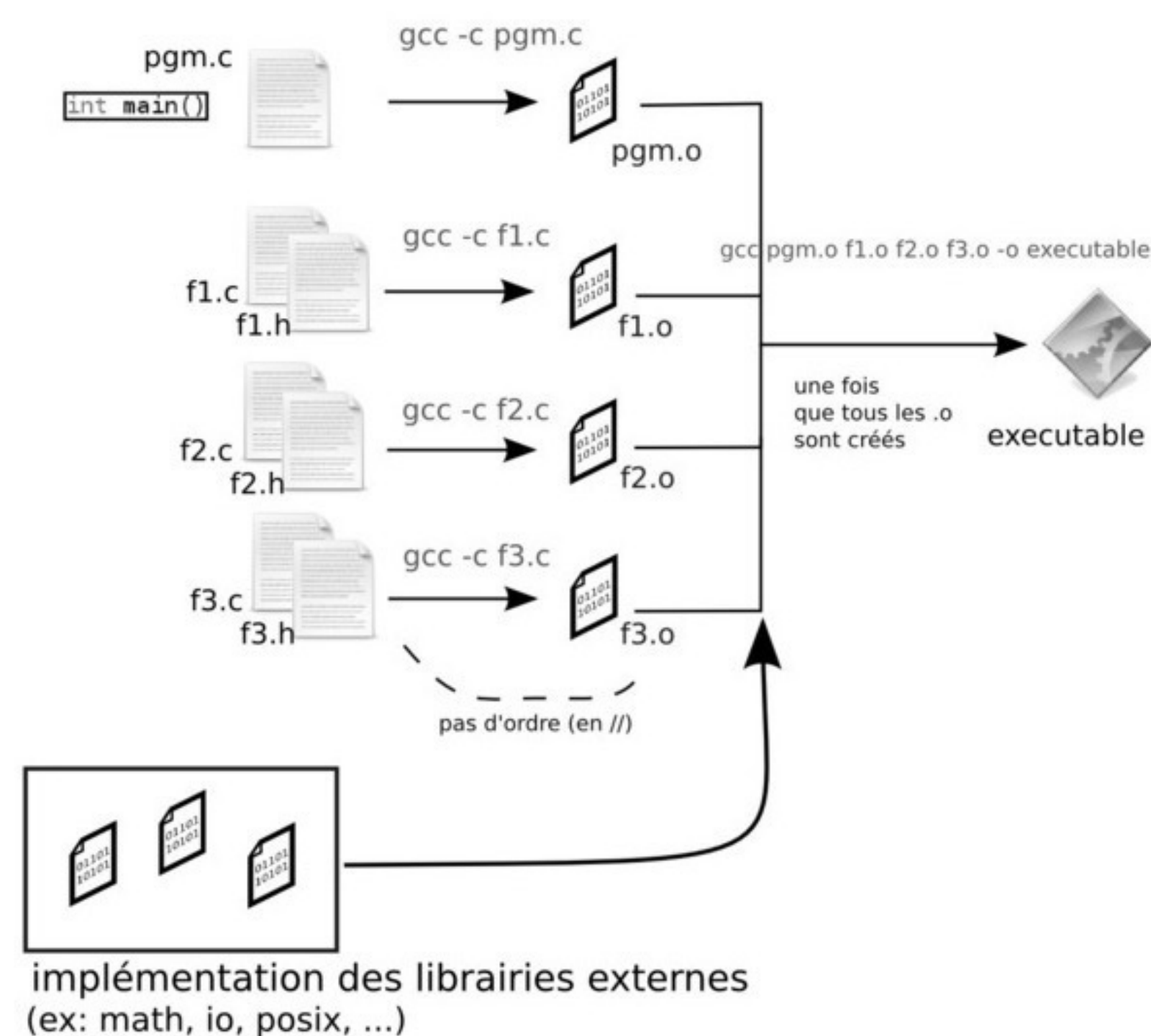
Chaine de compilation

Compilateur
Warnings
Compilation séparée
Préprocesseur
Makefile

→ **Edition de liens et bibliothèques**

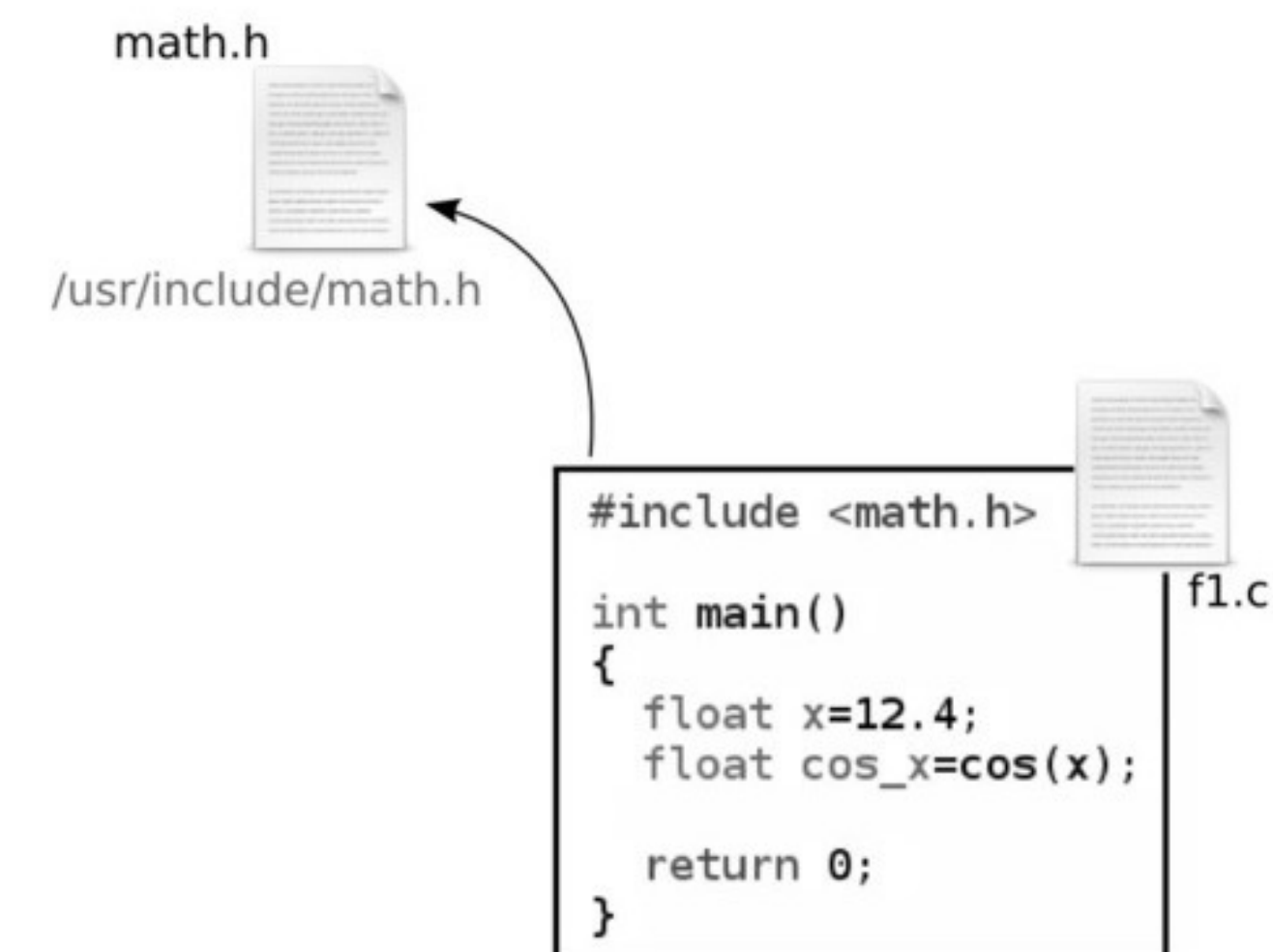
065

Edition de liens et bibliothèques



066

Edition de liens et bibliothèques



\$ gcc -c f1.c -o f1.o -Wall -Wextra -g
> OK

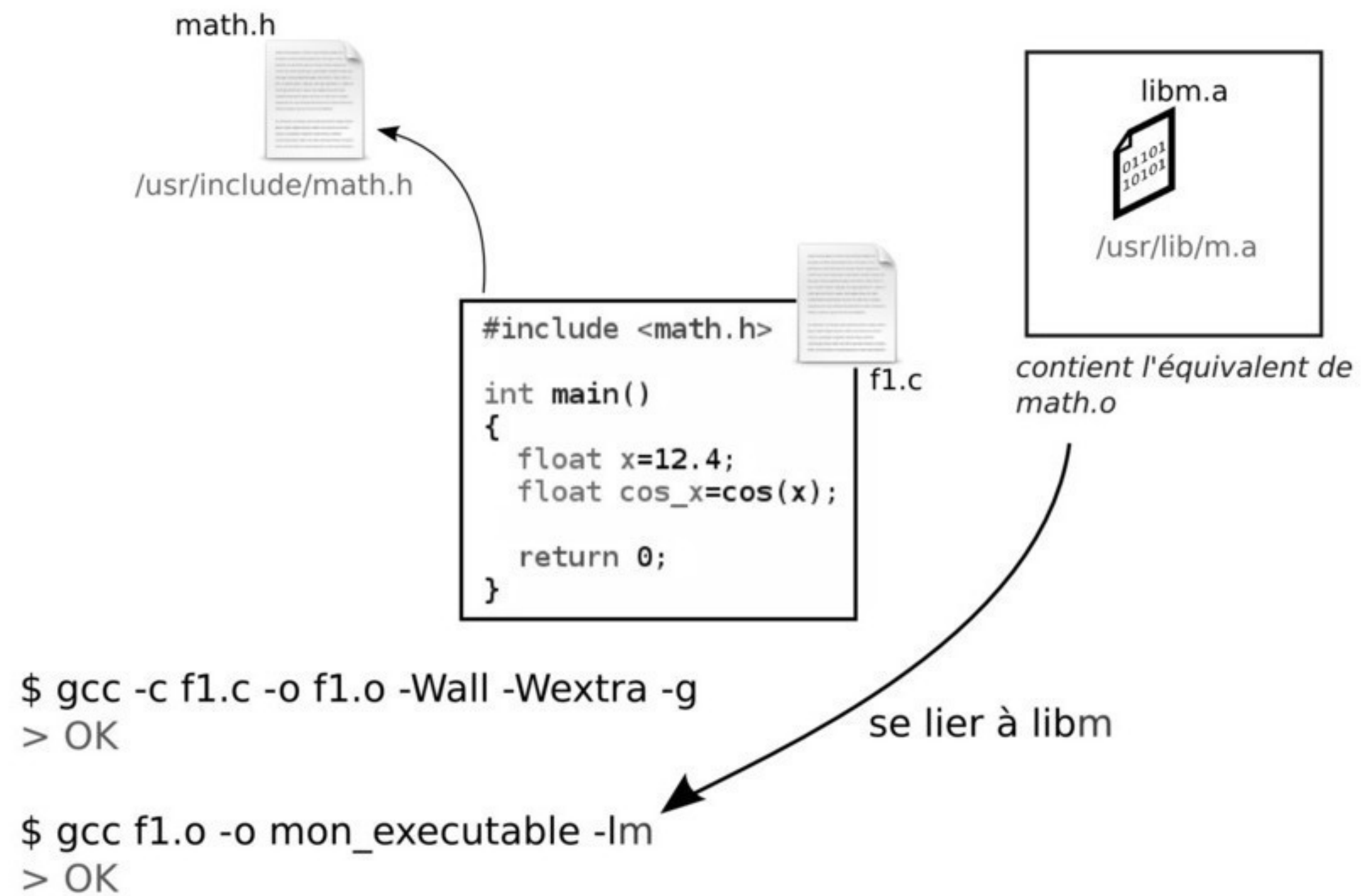
\$ gcc f1.o -o mon_executable
f1.o: In function `main':
f1.c:(.text+0x1a): undefined reference to `cos'
collect2: error: ld returned 1 exit status

Edition de lien échoue

Ne trouve pas
l'implémentation de cos

067

Edition de liens et librairies



068

Edition de liens et librairies

Librairie statique:

lib<NOM>.a équivalent à



Lors de l'édition de liens:

gcc fichier.o -l<NOM> équivalent à gcc fichier.o f1.o f2.o f3.o f4.o

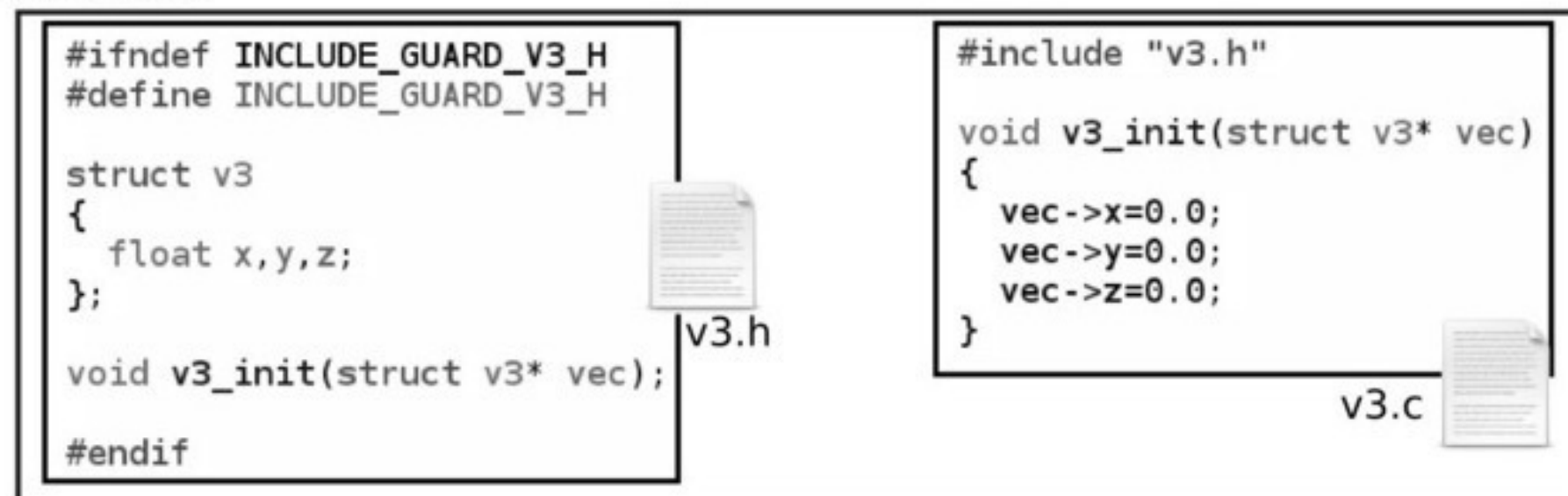
=> Inclusion des binaires dans l'exécutable finale

069

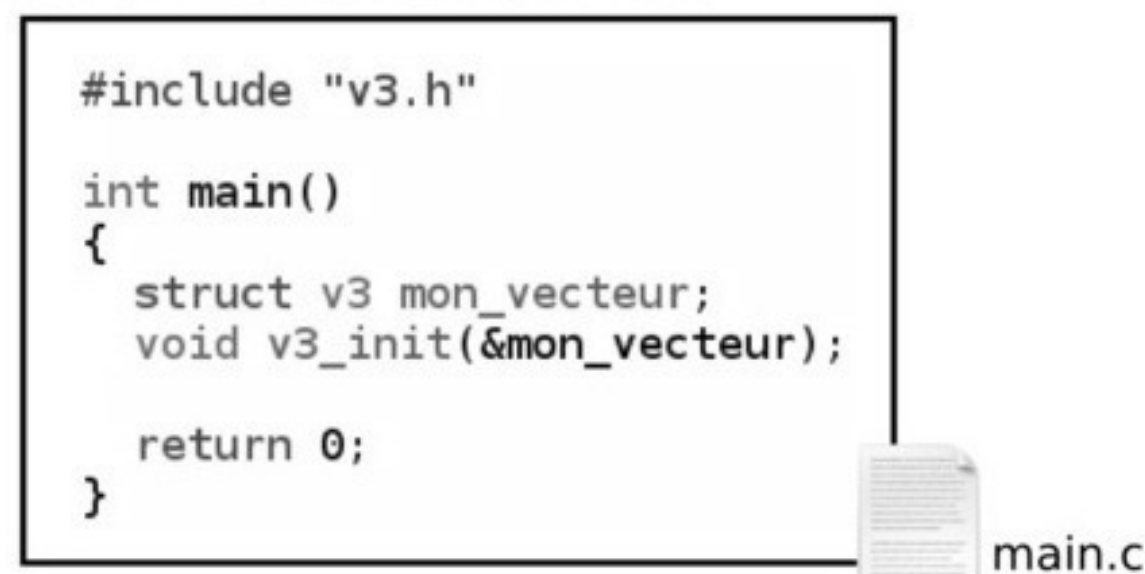
Edition de liens et librairies

Exemple de librairie:

librairie vecteur:



programme utilisant la librairie:

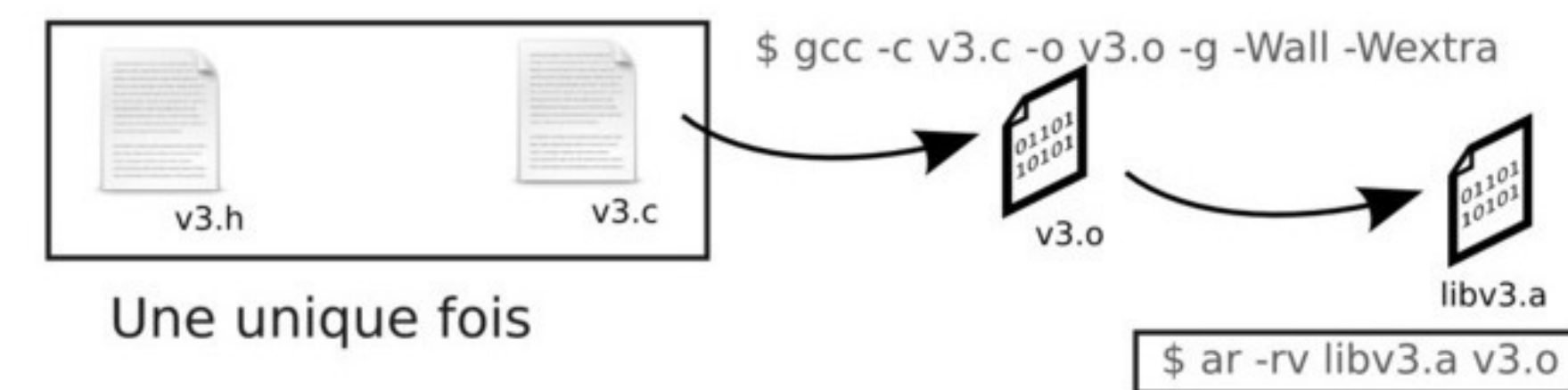


070

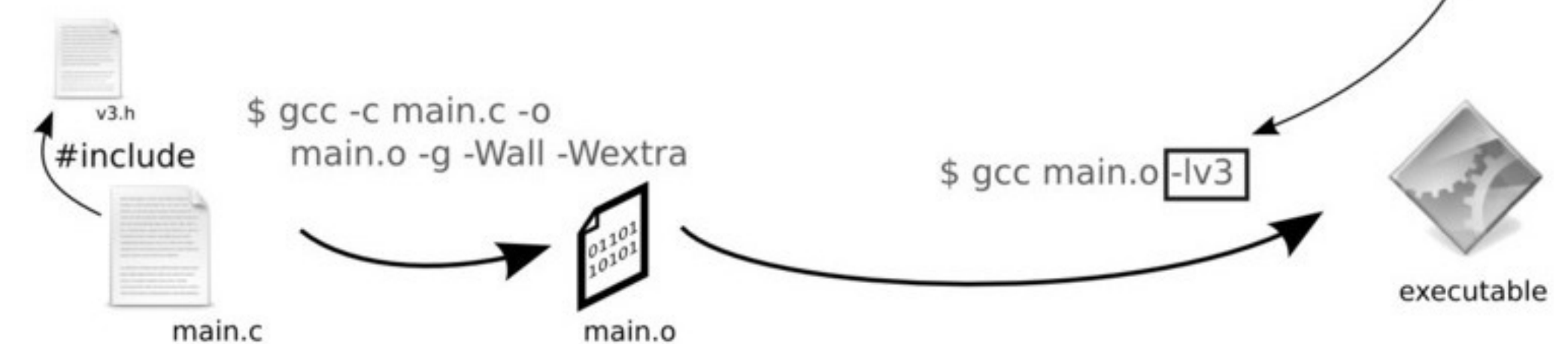
Edition de liens et librairies

Exemple de librairie:

1. Création de la librairie:



2. Utilisation de la librairie:



071

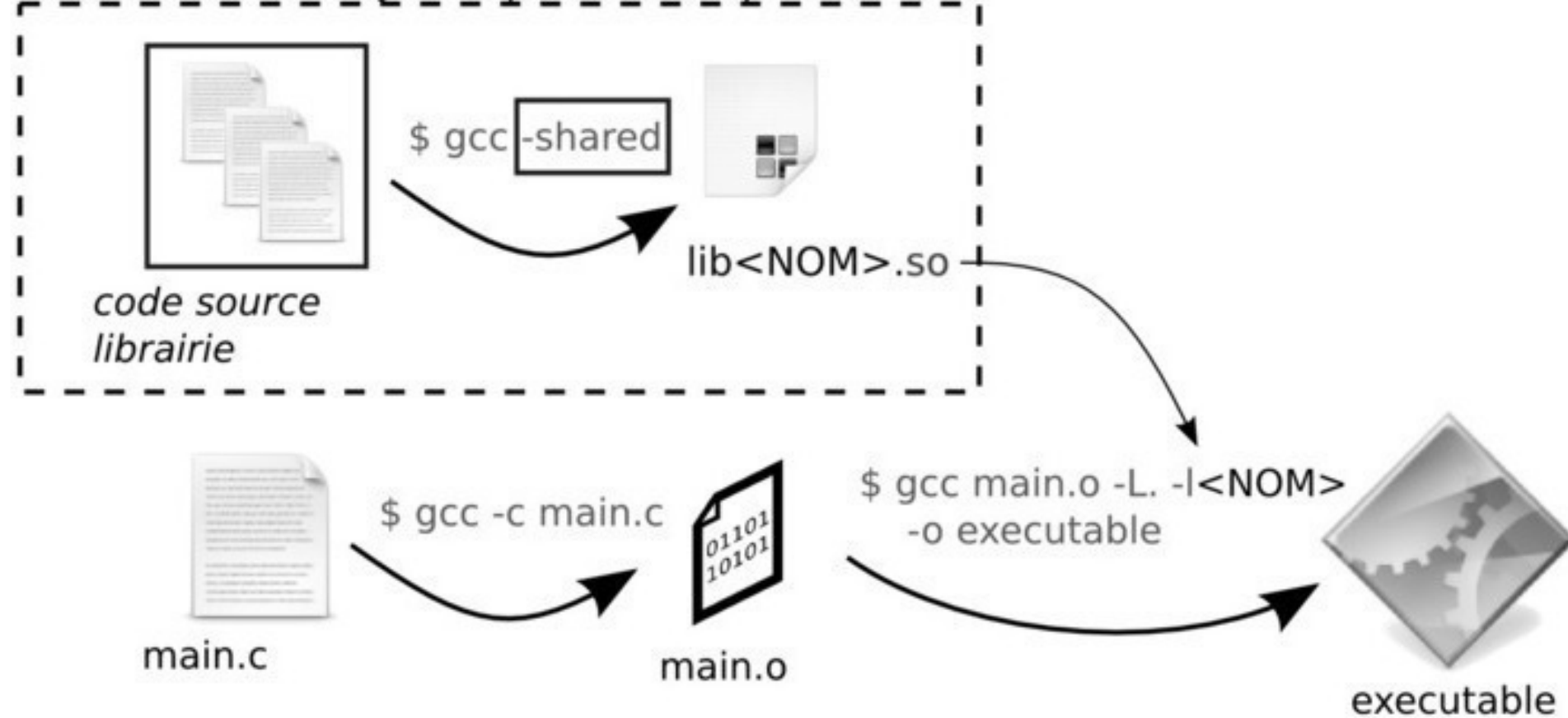
Edition de liens et librairies

Librairies dynamiques:


lib<NOM>.so

Sont chargées dynamiquement à **chaque execution**

creation de la lib dynamique: une unique fois



072

Edition de liens et librairies

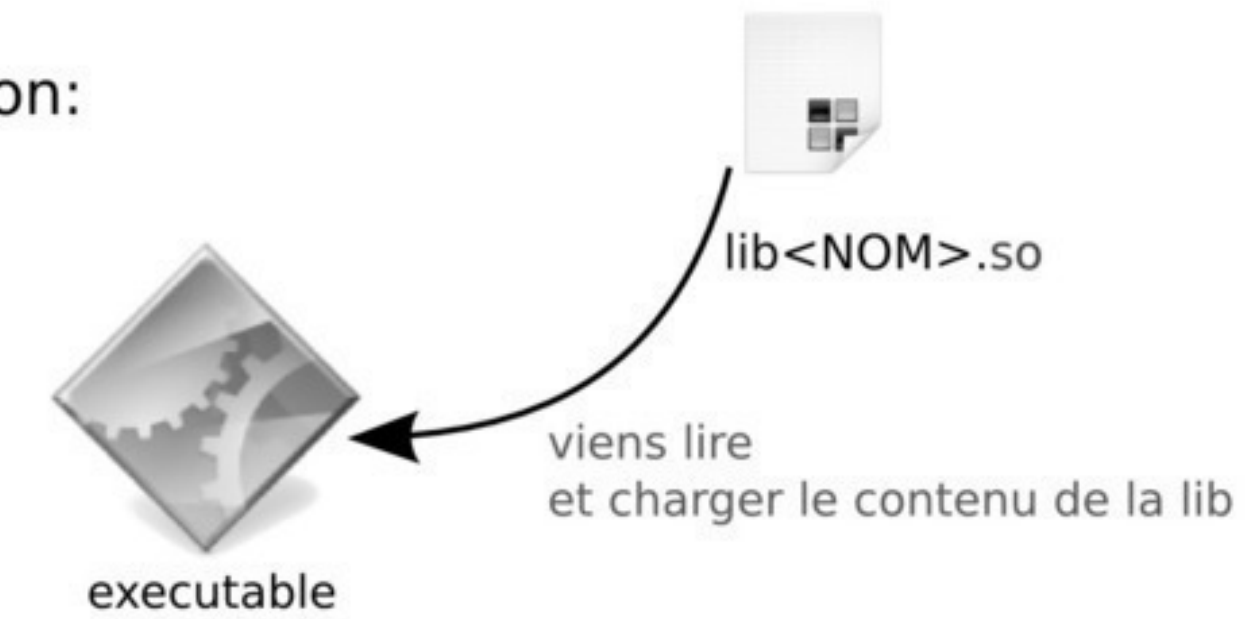
Librairies dynamiques:


lib<NOM>.so

Sont chargées dynamiquement à **chaque execution**

Lors de l'execution:

\$./executable



073

Edition de liens et librairies

Librairies dynamiques:


lib<NOM>.so

Sont chargées dynamiquement à **chaque execution**

Remarques:

1. Si la lib change => l'executable change de comportement
Sans avoir à être recompilée
(update, MAJ, comportement, ...)

2. Si la lib disparaît/est corrompu/n'est pas trouvée
=> l'executable ne peut pas être lancé

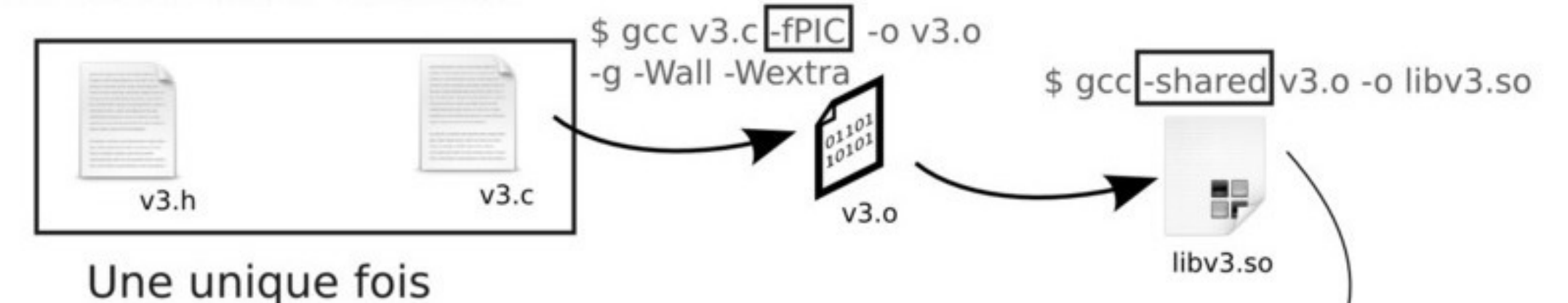
074

Edition de liens et librairies

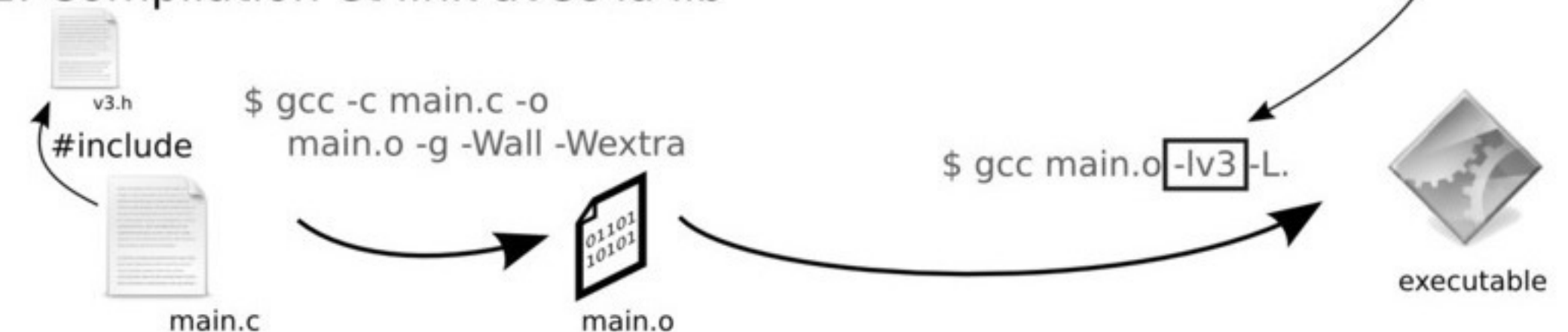
Exemple concret de librairies dynamiques:


libv3.so

1. Création de la librairie:



2. Compilation et link avec la lib



075

Edition de liens et librairies

Exemple concret de librairies dynamiques:



Utilisation de la librairie:



Par défaut: viens chercher la lib dans
/usr/lib/

\$ export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:.
ajout du repertoire local (.)

\$./executable

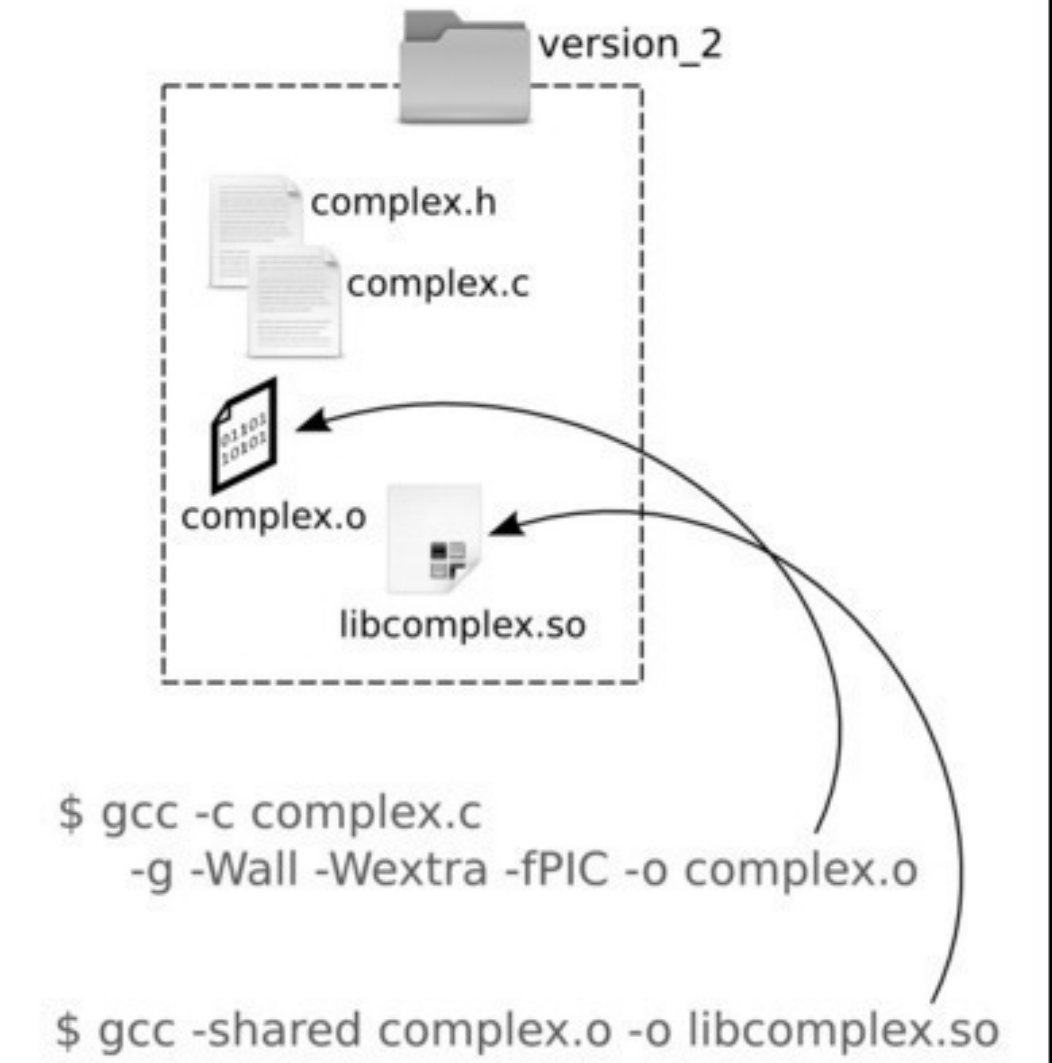
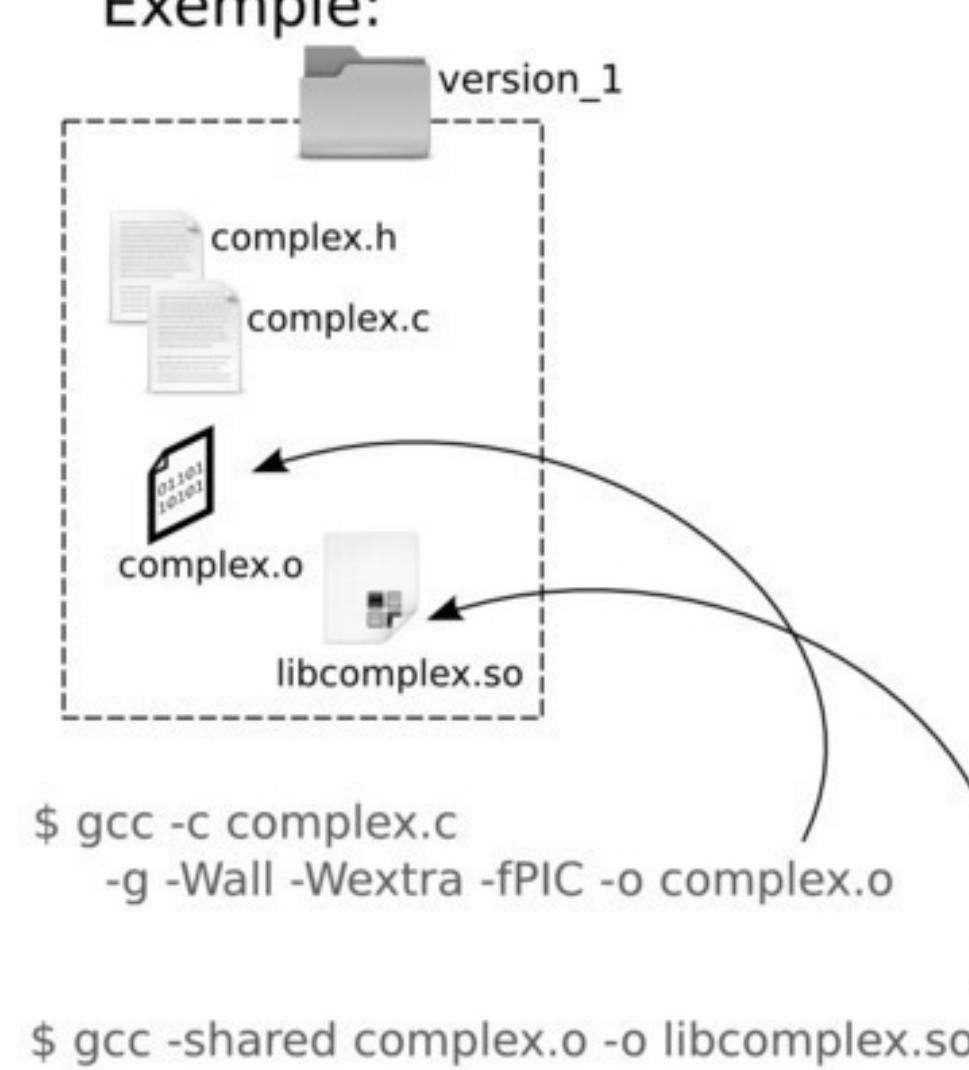
repertoires déjà enregistrés

repertoire courant

076

Edition de liens et librairies

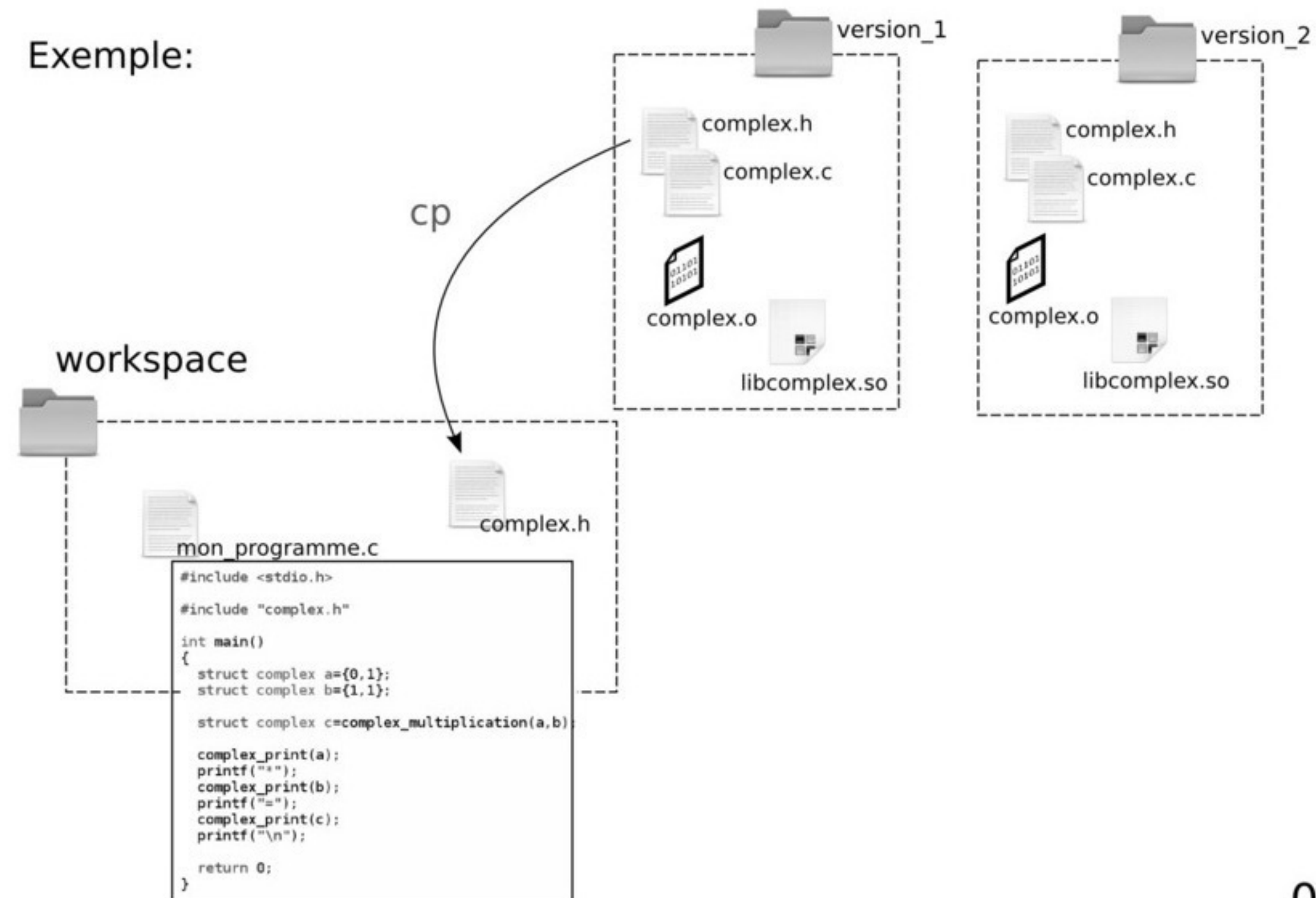
Exemple:



077

Edition de liens et librairies

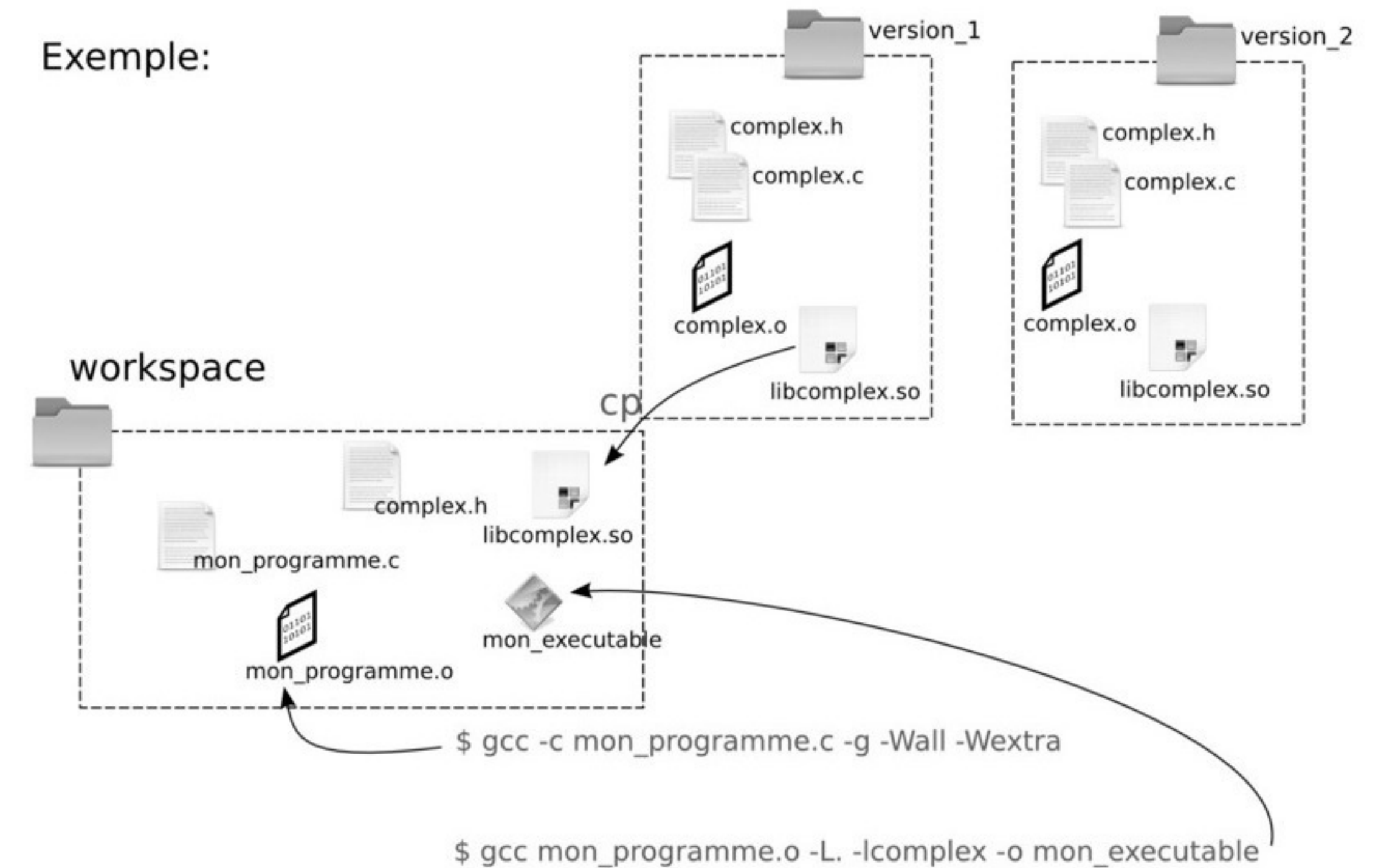
Exemple:



078

Edition de liens et librairies

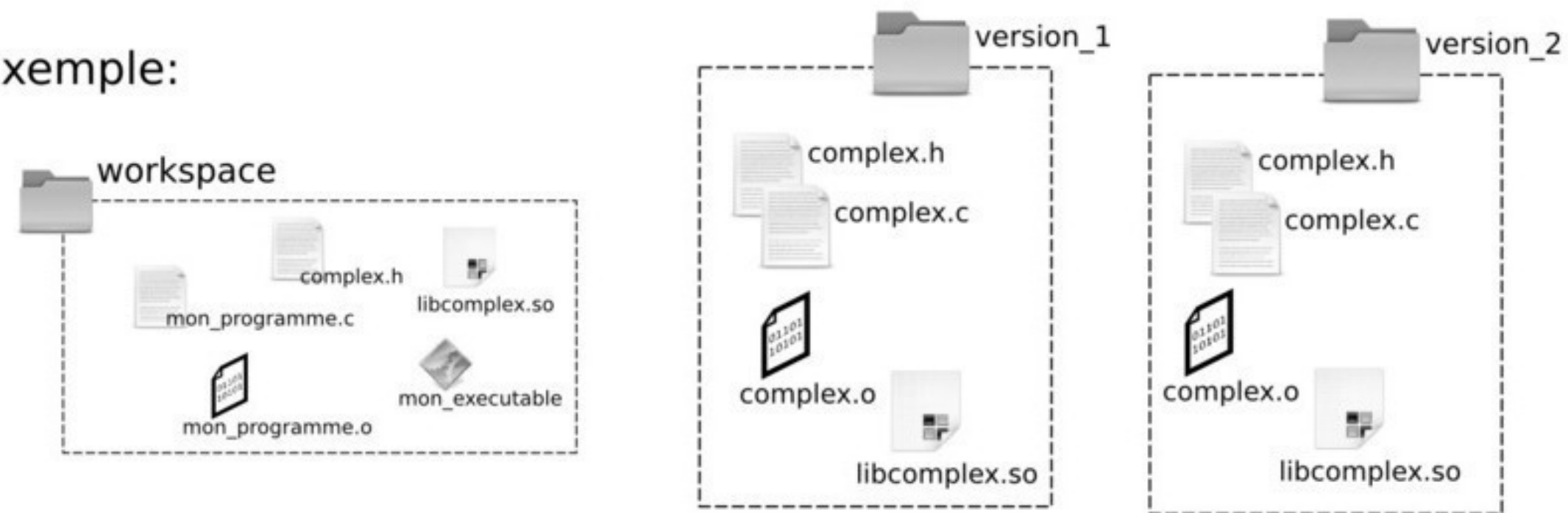
Exemple:



079

Edition de liens et librairies

Exemple:



```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
```

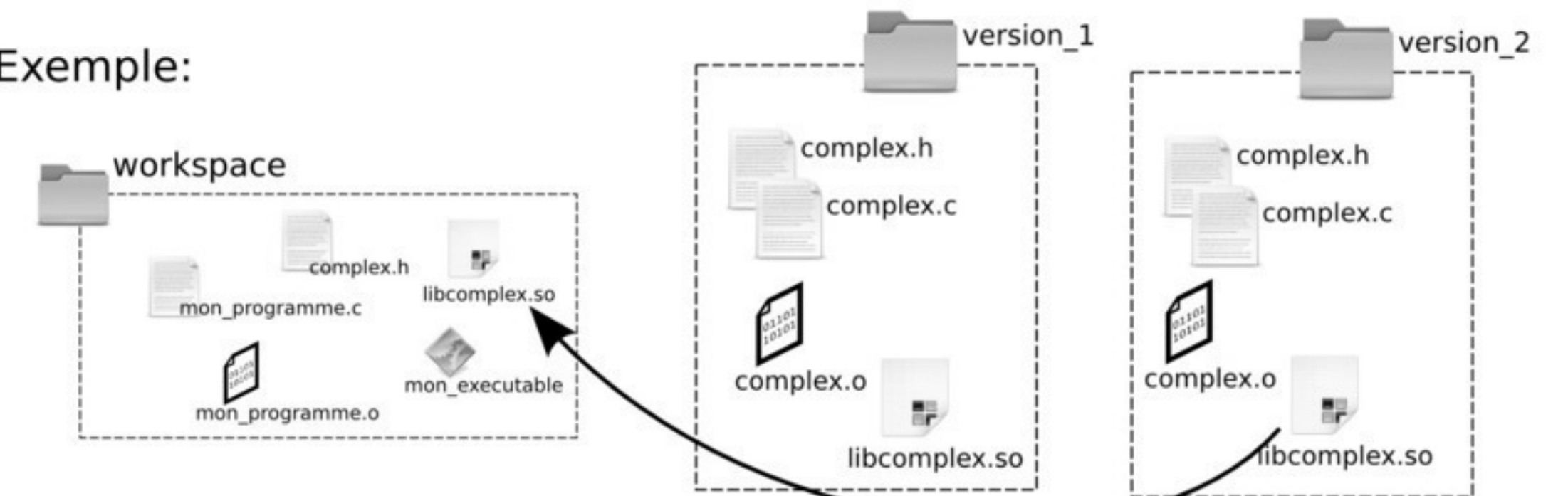
```
$ ./mon_executable
```

```
(0.000000+1.000000 i)*(1.000000+1.000000 i)=(1.000000+1.000000 i)
```

080

Edition de liens et librairies

Exemple:



cp
(erase version_1)

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
```

```
$ ./mon_executable
```

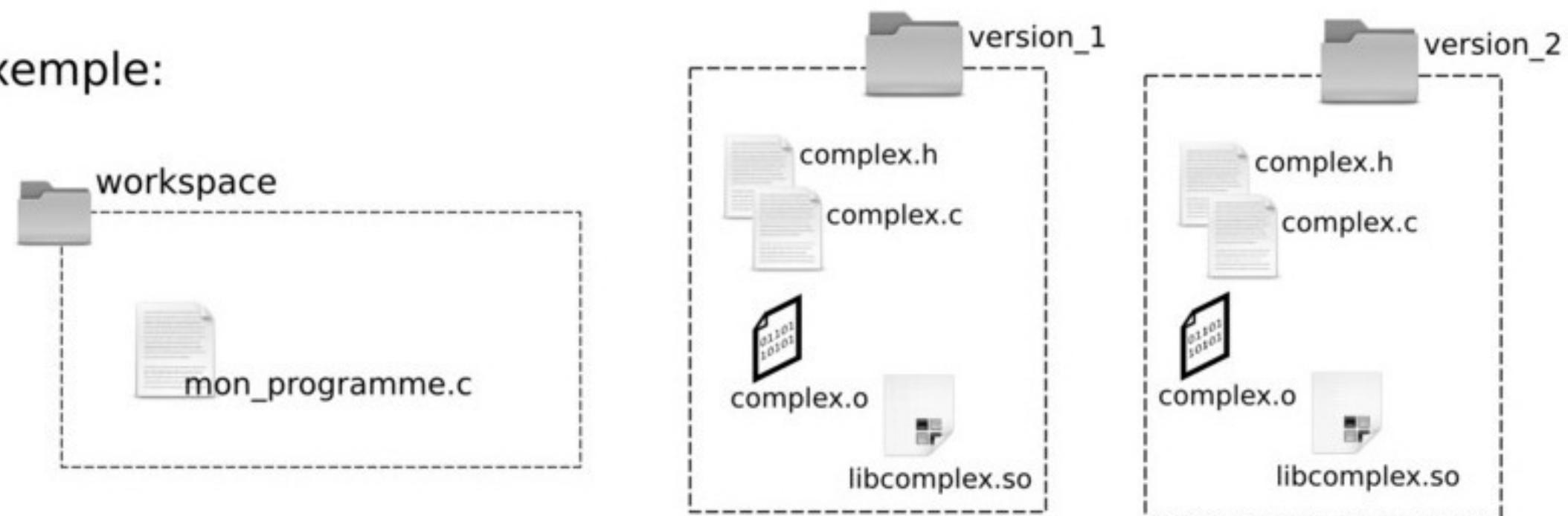
```
(0.000000+1.000000 i)*(1.000000+1.000000 i)=(-1.000000+1.000000 i)
```

(principe du patch logiciel)

081

Edition de liens et librairies

Exemple:



Si on ne veut pas copier/déplacer de fichiers:

```
$ gcc -c mon_programme.c -I version_1/ -g -Wall -Wextra
```

mon_programme.o

```
$ gcc mon_programme.o -L version_1/ -l complex -o executable
```

executable

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:version_1/
```

```
$ ./executable
```

082