Programmation en C
Cours 7

Licence Maths-Info
Aix-Marseille Université
2011-2012

Valentin Emiya
valentin.emiya@lif.univ-mrs.fr

12 mars 2012

Séances 7 et suivantes

Séances 1 à 6	PARTIEL	Séances 7 et suivantes
Apprentissage du langage Outils de base (compilation, etc.) Pratiques élémentaires		Pratique du C Bonnes pratiques

Objectifs des séances à venir :

- Bonnes pratiques et pratiques courantes pour le développement en C
- Mise en pratique « traitement d'image » (lecture/écriture, transformations/effets)

Séances à rattraper si possible :

mardis 20/03 et 17/04 à 16h (TP?)

Aujourd'hui

Pratique du langage C

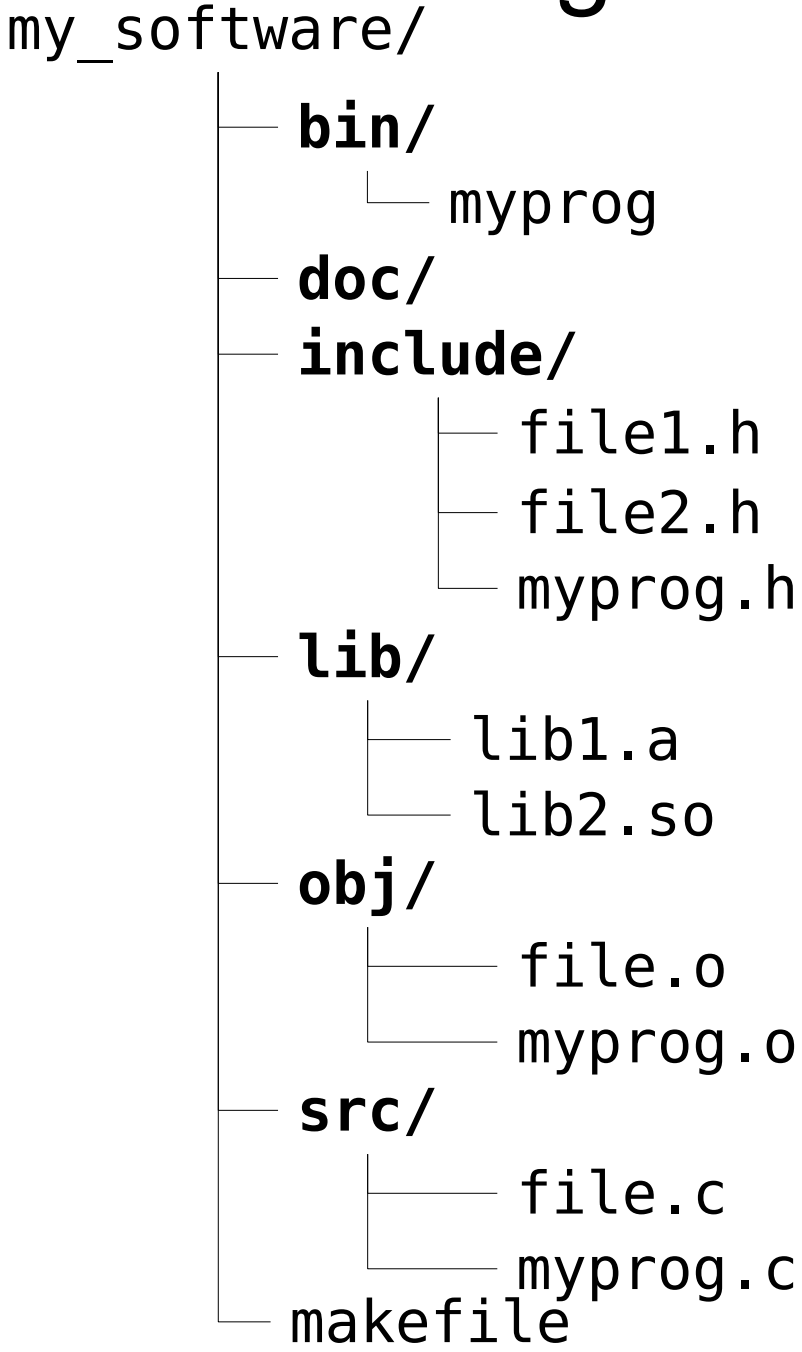
* Organisation des fichiers pour créer un logiciel
* Les bibliothèques statiques et dynamiques
  * Principe
  * Syntaxe, sémantique
  * Intérêts et inconvénients
  * Compilation
* Introduction du thème « traitement d'image »

Aujourd'hui

Pratique du langage C

* Organisation des fichiers pour créer un logiciel
* Les bibliothèques statiques et dynamiques
  * Principe
  * Syntaxe, sémantique
  * Intérêts et inconvénients
  * Compilation
* Introduction du thème « traitement d'image »

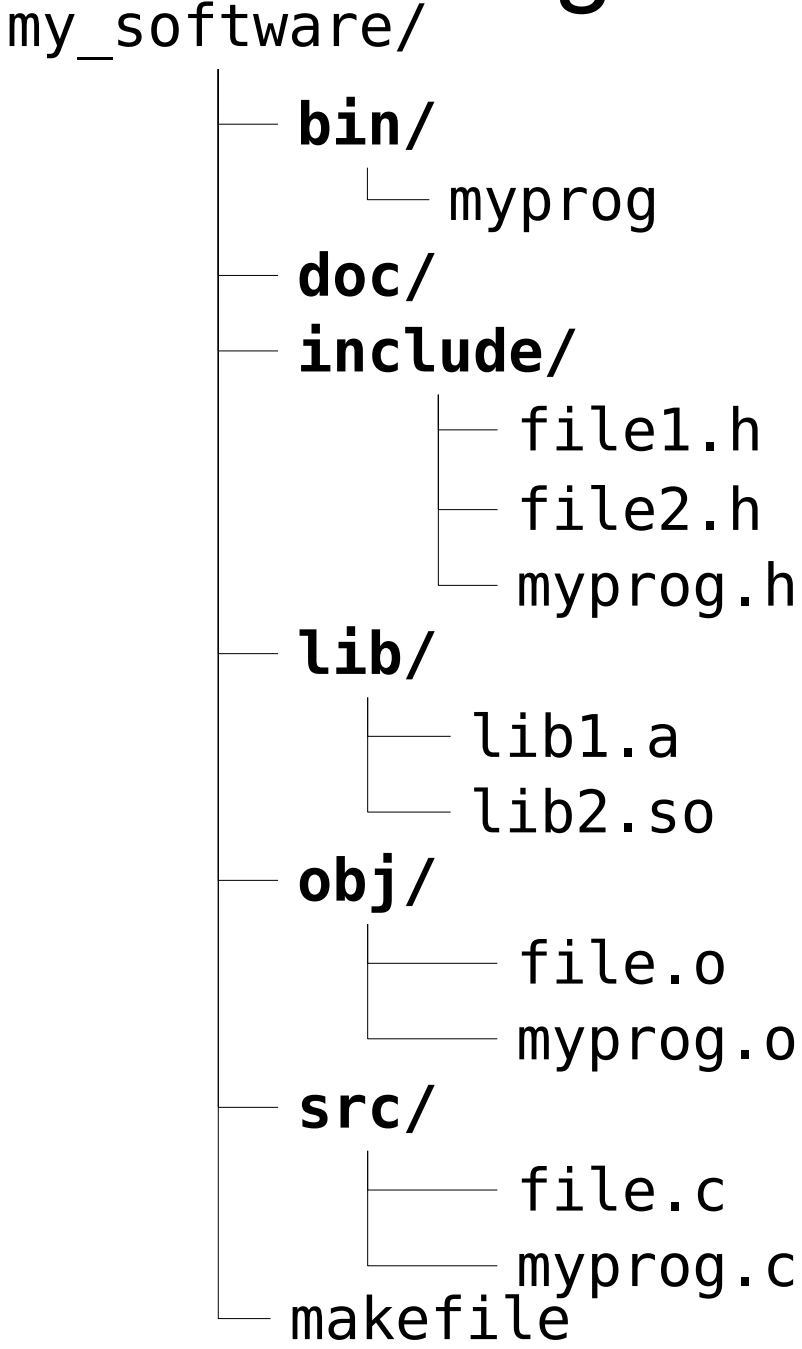
Organisation des fichiers



Une arborescence très répandue pour développer un logiciel :

- **`bin/`** : tous les exécutables
- **`doc/`** : la documentation
- **`include/`** : tous les fichiers en-tête (.h)
- **`lib/`** : toutes les librairies (.a, .so, ...)
- **`obj/`** : tous les fichiers objets (.o)
- **`src/`** : tous les fichiers source (.c)

Organisation des fichiers



Remarque :

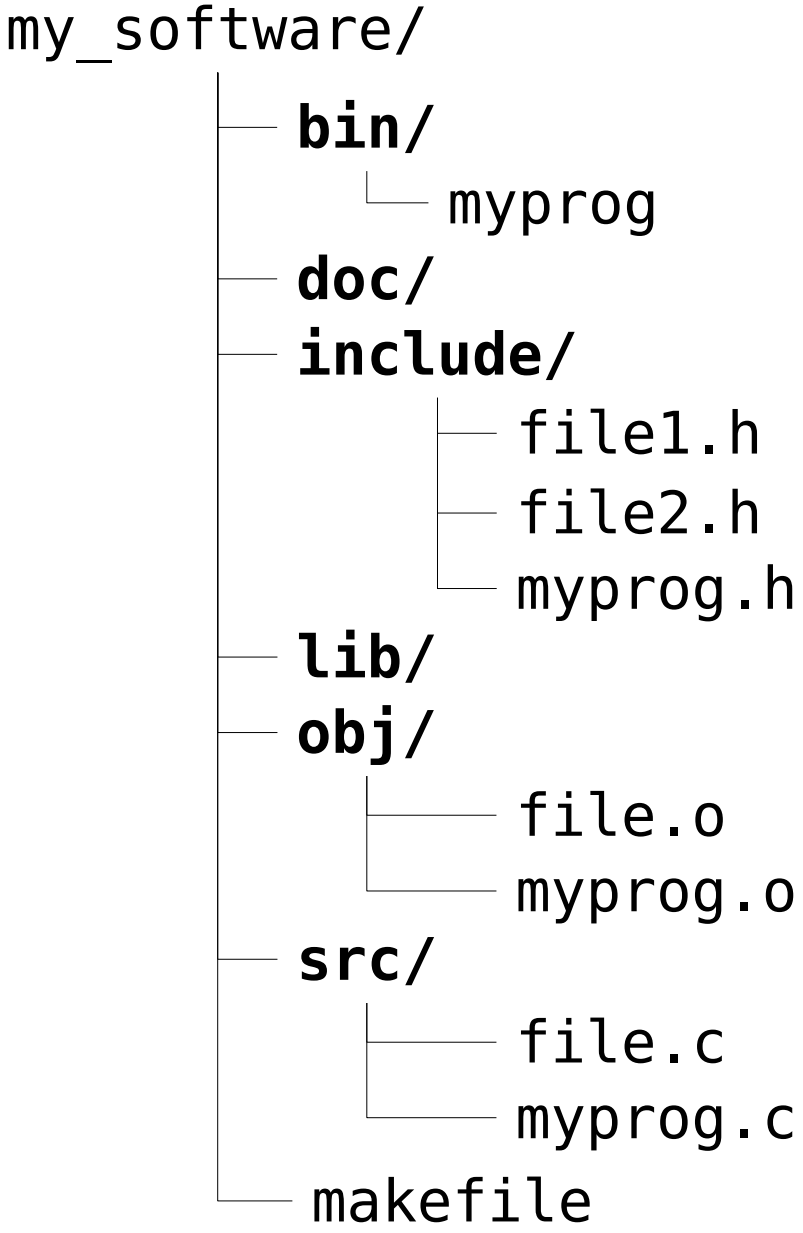
Il suffit de fournir include/, src/ et doc/ à l'utilisateur (ou au correcteur de TP)

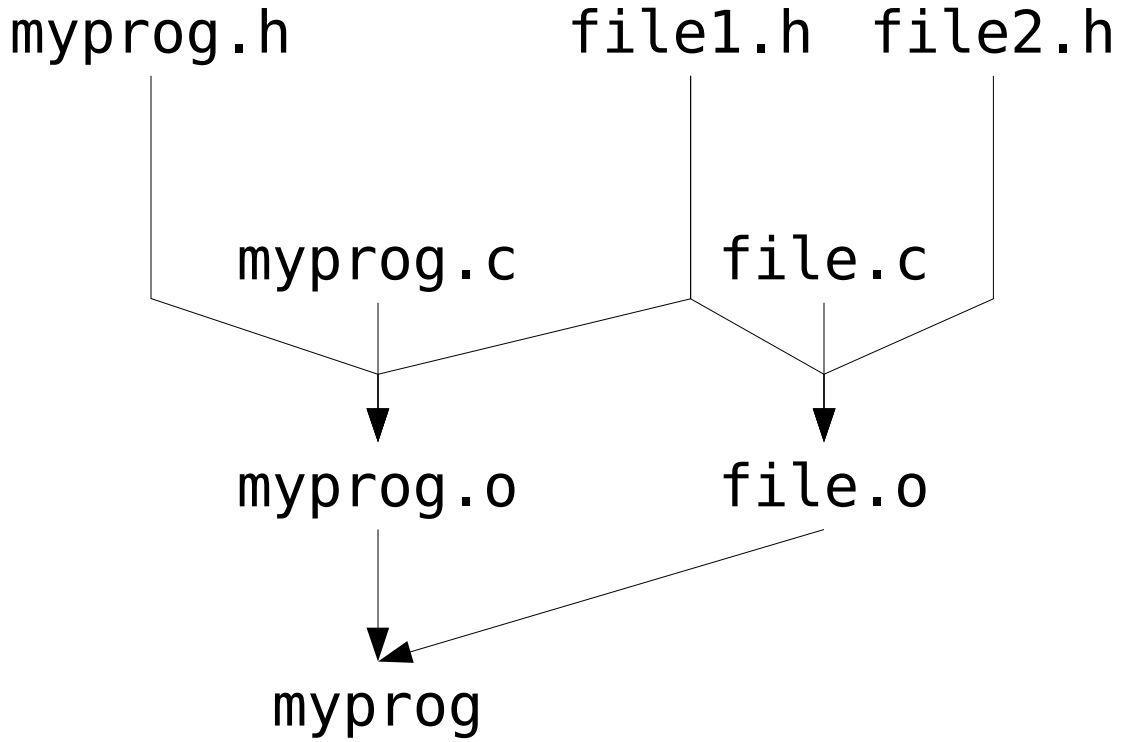
Organisation des fichiers

```
my_software/
├── bin/
│   └── myprog
├── doc/
├── include/
│   ├── file1.h
│   ├── file2.h
│   └── myprog.h
├── lib/
│   ├── lib1.a
│   └── lib2.so
├── obj/
│   ├── file.o
│   └── myprog.o
├── src/
│   ├── file.c
│   └── myprog.c
└── makefile
```

Vous pouvez adopter cette structure pour les TP à partir d'aujourd'hui

Organisation des fichiers





Prise en compte des sous-répertoires dans le makefile ?

Makefile

```
CC = gcc
BIN = ./bin/
INC = ./include/
OBJ = ./obj/
SRC = ./src/
CFLAGS = -W -Wall -ansi -pedantic -I $(INC)

exec: $(BIN)myprog

$(BIN)myprog: $(OBJ)myprog.o $(OBJ)file.o
   $(CC) $(CFLAGS) -o $@ $^

$(OBJ)myprog.o: $(SRC)myprog.c $(INC)file1.h
   $(CC) $(CFLAGS) -c $< -o $@

$(OBJ)file.o: $(SRC)file.c $(INC)file1.h $(INC)file2.h
   $(CC) $(CFLAGS) -c $< -o $@
```

Aujourd'hui

Pratique du langage C

* Organisation des fichiers pour créer un logiciel
* Les bibliothèques statiques et dynamiques
  * Principe
  * Syntaxe, sémantique
  * Intérêts et inconvénients
  * Compilation
* Introduction du thème « traitement d'image »

Les bibliothèques (*library*)

- Une bibliothèque contient la définition de fonctions pouvant être utilisées par plusieurs programmes.
- Une bibliothèque = regroupement selon une thématique.
- Une bibliothèque = ***un unique fichier objet*** issu d'un ou plusieurs fichiers sources.
- Exemples : voir dans `/usr/lib/`.
- Deux types de bibliothèques :

  statiques et **dynamiques**.

Bibliothèque statique vs. dynamique

Principe

- Bibliothèque statique : son code est inclus dans l'exécutable (édition de lien).
- Bibliothèque dynamique : son code n'est pas inclus dans l'exécutable, il doit être présent à l'exécution (utilisation d'un chargeur/*loader* pour le charger en mémoire en même temps que l'exécutable).

Bibliothèque statique : syntaxe

- Formats : librairie *toto*
  - `lib`*`toto`*`.a` (Unix, Linux),
  - `lib`*`toto`*`.lib` (Windows),
- Exemple : `libjpeg.a` est la bibliothèque pour manipuler les fichiers jpeg (images compressées)
- Remarque : le préfixe « lib » est important.
- Création : syntaxe

```
ar -rv libtoto.a file1.o file2.o file3.o
```

(`libtoto.a` regroupe le code de `file1.o, file2.o` et `file3.o`)

Bibliothèque statique : sémantique

= son code est inclus dans l'exécutable (similaire à ce que l'on a vu jusqu'à aujourd'hui)

- Avantage : exécutable autonome (*stand-alone*) et moins de problèmes de compatibilité.
- Inconvénient :
  - une MàJ de la bibliothèque n'est pas prise en compte dans l'exécutable sauf si recompilation
  - taille du fichier exécutable

Bibliothèque dynamique : syntaxe

= son code n'est pas inclus dans l'exécutable

- Formats : librairie *toto*
  - `lib`*`toto`*`.dll` (*Dynamic Link Library*, Windows),
  - `lib`*`toto`*`.so` (*shared object*, Unix, Linux),
  - `lib`*`toto`*`.dylib` (Mac OS X),
  - ...
- Remarque : le préfixe « lib » est important.
- Création :

```
gcc -o libtoto.so -shared file1.o file2.o file3.o
```

Bibliothèque dynamique : sémantique

= son code n'est pas inclus dans l'exécutable

- Avantage : une MàJ de la bibliothèque ne nécessite pas une recompilation de l'exécutable.
- Inconvénient : l'exécutable n'est pas autonome, possibles incompatibilités.

Où trouver/placer une bibliothèque

En général, plusieurs répertoires contiennent spéciquement des bibliothèques.

Par exemple (Unix, Max OSX) :

- `/usr/local/lib` pour les librairies utilisées par plusieurs utilisateurs
- `~/lib` pour les librairies d'un utilisateur particulier
- La variable d'environnement `LD_LIBRARY_PATH` permet de localiser les bibliothèques dynamiques.

Utiliser une bibliothèque

Comment compiler un programme qui utilise une bibliothèque ?

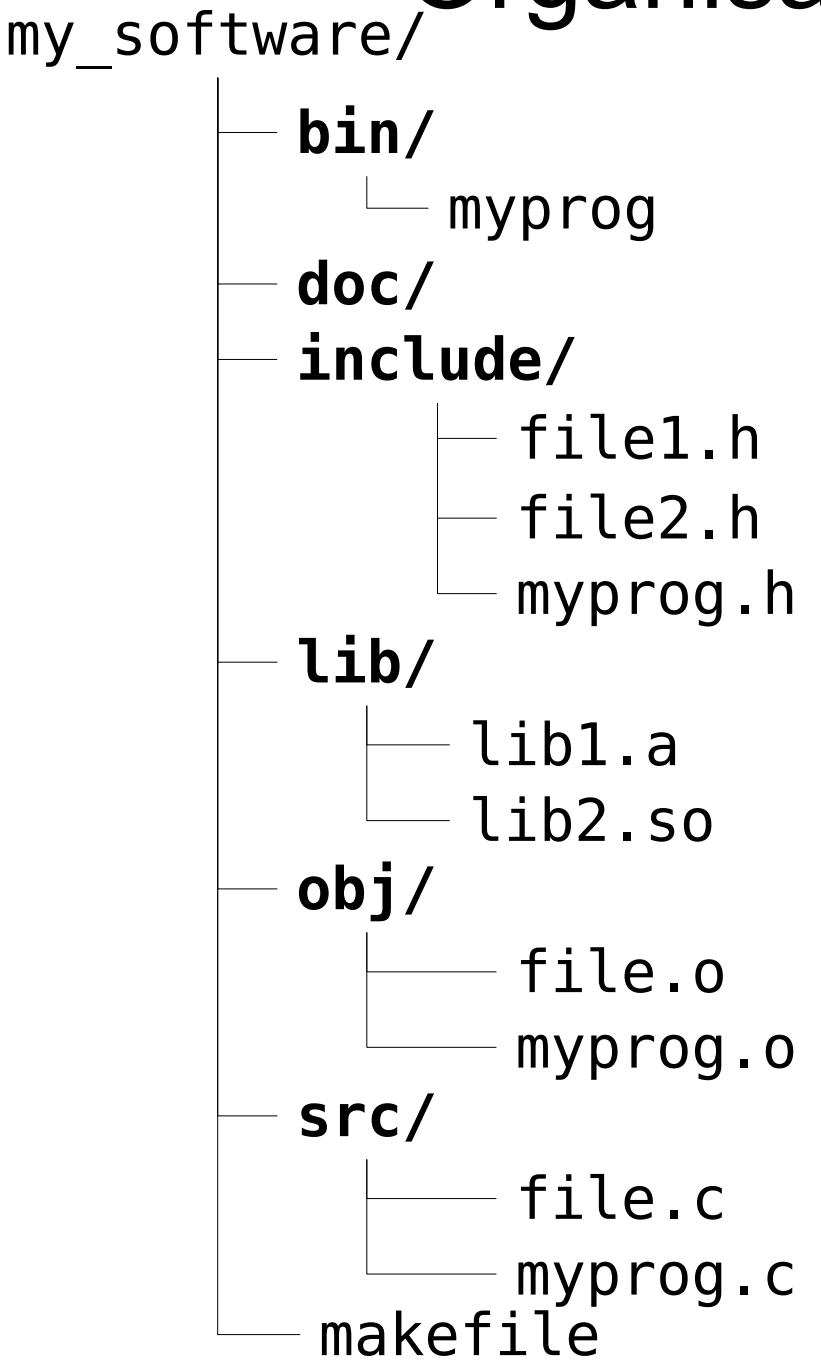
Exemple :

```
gcc -c file.c
gcc -c prog.c
gcc file.o prog.o -L/usr/lib -lm -o prog
```

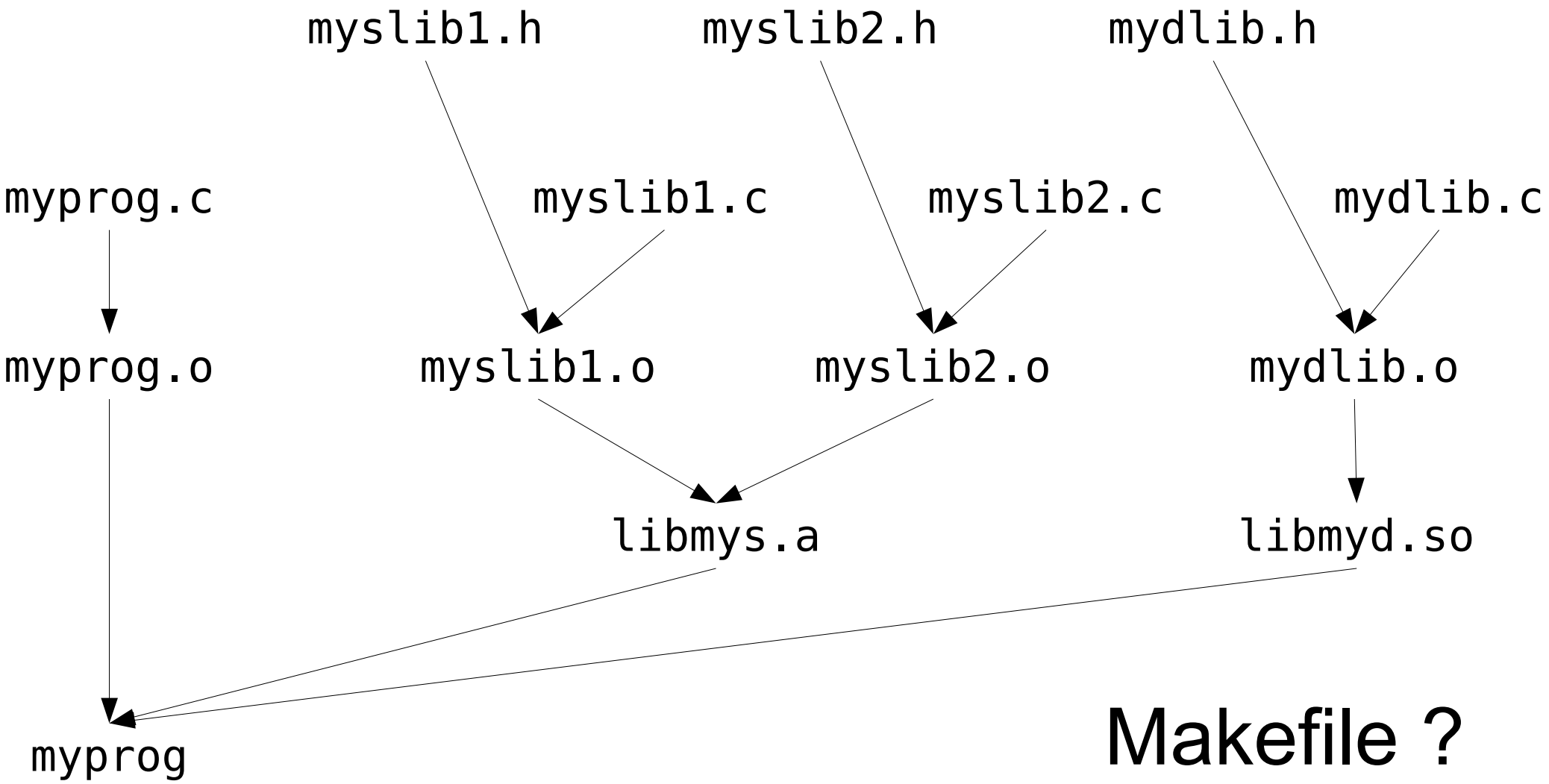
- option `-L` : répertoire où se trouvent les bibliothèques
- option `-l`*`xxx`* : utilisation de `libxxx.a` ou `libxxx.so` (enlever le préfixe « `lib` »)

NB : « -L » et « -l » comme *Library*

Organisation des fichiers



Compilation : exemple



Makefile ?

Makefile

```
CC = gcc
AR = ar -rv
BIN = ./bin/
INC = ./include/
LIB = ./lib/
OBJ = ./obj/
SRC = ./src/
CFLAGS = -W -Wall -ansi -pedantic -I$(INC) -L$(LIB)
exec: $(BIN)myprog
libs: $(LIB)libmys.a $(LIB)libmyd.so
$(BIN)myprog: $(OBJ)myprog.o
   $(CC) $(CFLAGS) -o $@ $^ -lmys -lmyd
$(LIB)libmys.a: $(OBJ)myslib1.o $(OBJ)myslib2.o
   $(AR) $@ $^
$(LIB)libmyd.so: $(OBJ)mydlib.o
   $(CC) -o $@ -shared $^
$(OBJ)myprog.o: $(SRC)myprog.c $(INC)myslib1.h
   $(CC) $(CFLAGS) -c -o $@ $<
$(OBJ)myslib1.o: $(SRC)myslib2.c $(INC)myslib1.h
   $(CC) $(CFLAGS) -c -o $@ $<
$(OBJ)myslib2.o: $(SRC)myslib2.c $(INC)myslib2.h
   $(CC) $(CFLAGS) -c -o $@ $<
$(OBJ)mydlib.o: $(SRC)mydlib.c $(INC)mydlib.h
   $(CC) $(CFLAGS) -c -o $@ $<
```

Aujourd'hui

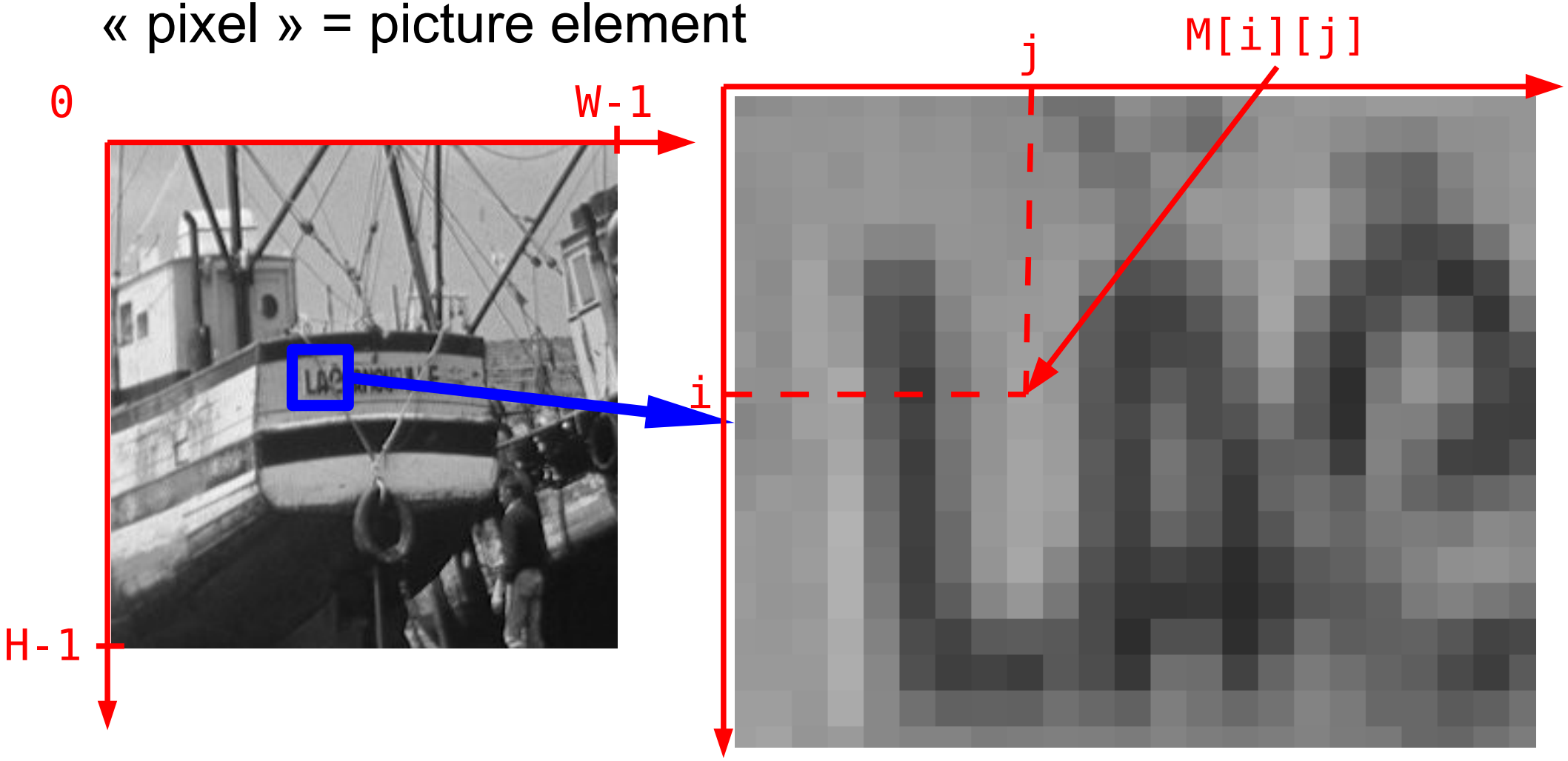
Pratique du langage C

* Organisation des fichiers pour créer un logiciel
* Les bibliothèques statiques et dynamiques
  * Principe
  * Syntaxe, sémantique
  * Intérêts et inconvénients
  * Compilation
* Introduction du thème « traitement d'image »

Image numérique

Une image numérique = une matrice M (HxW) de pixels

« pixel » = picture element



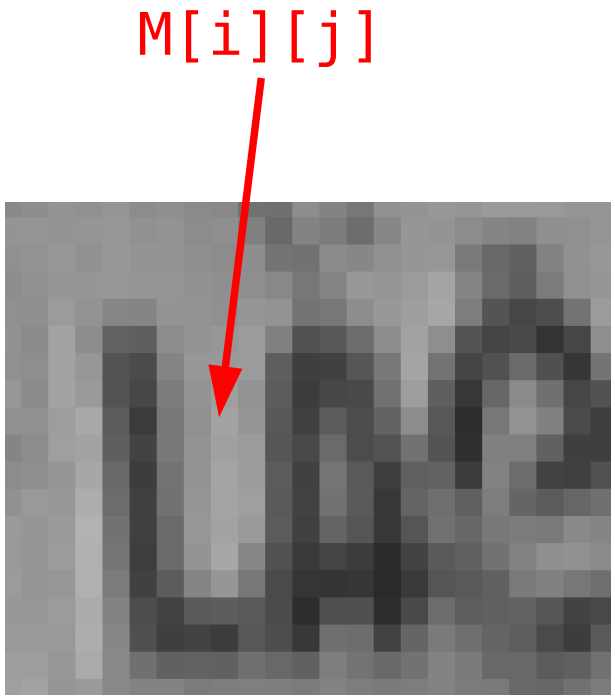
Images en niveaux de gris

Selon les formats utilisés, les valeurs de `M[i][j]` sont

- soit des entiers entre 0 et une valeur max (par exemple 255)
- soit des flottants entre 0 et 1

Par convention,

- 0 = noir
- valeur max = blanc



Images en couleur

`M[i][j]` est un triplet (r,g,b) de composantes rouge, vert et bleu qui sont

- soit des entiers entre 0 et une valeur max (par exemple 255)
- soit des flottants entre 0 et 1

Ces composantes sont les couleurs primaires en synthèse additive.

NB : il existe d'autres façons représenter les images

Formats de fichiers

- Très nombreux formats existants : png, pgm, jpg, bmp, eps, tiff, gif, etc.
- Format pgm (*Portable GrayMap*) : format simple où l'on stocke directement la matrice `M` (pas de compression) sous forme ASCII (texte) ou binaire

Fichier pgm ASCII

Nombre magique P2 et caractère d'espacement

Largeur de l'image et caractère d'espacement

Hauteur de l'image et caractère d'espacement

Valeur maximale utilisée (doit être inférieure à 65536) et caractère d'espacement

Données ASCII de l'image :
 L'image est codée ligne par ligne en partant du haut
 Chaque ligne est codée de gauche à droite
 Chaque pixel est codé par une valeur en caractères ASCII, précédée et suivie par un caractère d'espacement.
 Aucune ligne ne doit dépasser 70 caractères.

Toutes les lignes commençant par # sont ignorées.

Fichier pgm ASCII

Exemple :

```
P2
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0  0  0  0 0 15 0  0  15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0  0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0  0  0  0 0 15 0  0  0  0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0  0  0  0
0 0 0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0
```

Fichier pgm binaire

Idem sauf que

- Le nombre magique est P5
- Les données sont écrites en mode binaire (pas en mode texte) sur 1 ou 2 octets (selon que la valeur maximale est inférieure à 255 ou pas)

Avantage : gain de place

Remarque : il existe le même genre de formats de fichiers pour les images noir et blanc (pbm, *portable bitmap*) et couleur (ppm, *portable pixmap*).

Traitement d'image

A travers plusieurs séances de TP, nous verrons :

- Lecture / écriture de fichiers pgm
- Traitements/effets sur les images :
  - Effets de base (familiarisation)
  - Détection de contours (manipulation tableaux)
  - Débruitage, transformations via traitements non-linéaires (passage de fonction en argument)

Aujourd'hui

Pratique du langage C

* Organisation des fichiers pour créer un logiciel
* Les bibliothèques statiques et dynamiques
  * Principe
  * Syntaxe, sémantique
  * Intérêts et inconvénients
  * Compilation
* Introduction du thème « traitement d'image »