# Modules, etc

# Modules

**Principe** : En général, un module C est défini par un couple de fichier *module.h* et *module.c*, tel que :

- <name> est le nom du module
- <name>.h est le fichier *header* avec la déclaration des fonctions publiques du module + leur documentation… (interface ou API)
- <name>.c est le fichier avec la définition des fonctions du modules, les fonctions publiques déclarée dans le *header*, mais aussi les fonctions auxiliaires privées du module (à définir comme *static*)

**Distribution** : L'usage veut que l'on distribue un module (ou plusieurs) sous la forme d'une bibliothèque réutilisable, tel que :

- <name>.a, est le fichier de la bibliothèque statique au format binaire (x86_64)
- <name>.h  est le fichier qui contient la déclaration des fonctions publique du module, que l'on souhaite utiliser dans son code (#include "<name>.h")

# Modules

**Utilisation d'un type opaque pour masquer la définition d'une structure…**

- Le module définit et manipule un *objet* dont l'état est sauvegardé dans une instance d'un type *struct data_s* (allouée dynamiquement avec *malloc*)
- Le type *struct* est défini dans le fichier <name>.c et est déclaré dans le fichier <name>.h (*forward declaration*)
- Manipulation de la structure dans l'interface du module <name>.h uniquement à l'aide d'un pointeur (*struct data_s* *) → taille du type pointeur connue par le compilateur !
- Utilisation du pointeur constant (*const struct data_s* *) pour interdire de modifier le contenu de la structure (*readonly vs readwrite access*)
- …

# Modules : Exemple *foobar*

Reprenons l'exemple de *foobar*, avec une bibliothèque *libfb.a* qui est composée de deux modules :

- Module *foo* : *foo.h & foo.c*
- Module *bar* : *bar.h & bar.c*

**Démo**

- Sources : https://pt2.gitlabpages.inria.fr/support/site/td01/misc/foobar.zip
- Makefile (cf. correction)

# Modules : Modèle

**/* module.h */**

```
#ifndef __MODULE_H__
#define __MODULE_H__

/* required by declaration */
#include <stdbool.h>

/* opaque data type */
typedef struct data_s *data;
typedef const struct data_s *cdata;

/* declaration */
bool odd(cdata x); // accessor
void inc(data x);  // modificator

#endif /* __MODULE_H__ */
```

**/* module.c */**

```
/* required for implementation */
#include <stdbool.h>
#include <stdio.h>

/* include & check declaration */
#include "module.h"

/* data type definition */
struct data_s {
  int val;
};

/* private functions (static) */
static int get(cdata x) { return x->val; }
static void set(data x, int val) { x->val = val ; }

/* public functions */
bool odd(cdata x) { return get(x) % 1; }
void inc(data x) { set(x, x->val +1 ); }
```

# Modules

**Problème de l'inclusion multiple de *header*, qu'il faut protéger !**

```
// module.h
#ifndef __MODULE_H__
#define __MODULE_H__
// déclarations …
#endif

// file1.h
#include "module.h"
…

// file2.h
#include "module.h"
…

// main.c
#include "file1.h"
#include "file2.h" // attention, double inclusion de module.h !
…
```

# Interface Texte

# TD02 : Interface en Mode Texte

Objectifs : développer une interface en mode texte en s'appuyant sur une bibliothèque

- Bibliothèque *game*
  - Interface : *game.h* + *game_aux.h*
  - Implémentation : fichier *libgame.a*
- Interface de Programmation (API)
  - Doc. https://pt2.gitlabpages.inria.fr/support/doc/html/
  - Notion de type opaque *game* & *cgame*
- Programme *game_text* (*game_text.c*)
  - Un exécutable, point d'entrée dans la fonction *main()*
  - Utilisation des fonctions de la bibliothèque → ne pas réinventer la roue, mais lire attentivement la doc !!!
  - Lire l'entrée standard avec *scanf()* & écrire sur la sortie standard *printf(), game_print(), …*
  - Tester son programme avec des redirections dans le *shell* :

```
$ echo "w 0 0 q" | ./game_text
$ cat moves.txt | ./game_text
```

# A Propos de *scanf*

```
// chaînes de caractèred formatées
int x = 10;
printf("value = %d\n", x);

// Important: scanf() input specifiers like "%d" (to match integer),
// "%u" (to match unsigned integer), "%f" (to match float), %s (to match word), ...
// generally ignore leading spaces, except: "%c" (to match char).
// So, you need to use " %c" to skip explicitly leading white-space
// (space, tab, newline, ...).

int n = 0;
int r = scanf("%d", &n);
if (r == 1) printf("Matching success of integer %d.\n", n);
```

⇒https://pt2.gitlabpages.inria.fr/support/site/td02/misc/demo-scanf.c

# Codons un petit jeu pour s’entraîner !

**Deviner un nombre tiré au hasard entre 0 et 99…**

⇒ https://pt2.gitlabpages.inria.fr/support/site/td02/misc/guess.c

# Du Code Propre !

**Voici les objectifs généraux qui définissent un excellent projet :**

- rendre un code fonctionnel, sans duplication de code, ni bugs (compilation sans warning a minima), ni fuite mémoire ;
- rendre un code interopérable, qui respecte rigoureusement les interfaces définies ;
- rendre un code suffisament testé, afin d'éviter toute regression fonctionnelle lors de son cycle de vie ;
- rendre du code propre et lisible (formatage automatique, nommage des variables et fonctions, pas de constantes magiques, langage anglais, …)
- produire du code suffisament commenté, afin de faciliter sa prise en main par d'autres développeurs, sa maintenance et son évolution vers de nouvelles fonctionnalités, …

**Un excellent résumé par N. Bonichon…**

⇒ https://mediapod.u-bordeaux.fr/video/12904-commenter-et-nettoyer-son-code