

# Programmation modulaire

Un programme en langage C peut contenir plusieurs dizaines de milliers d'instructions !

Même si on fait un découpage en fonctions, il n'est pas recommandé de ne faire qu'un seul fichier source, car :

- dur à maintenir (mais où se trouve donc cette #@!?! de fonction ?)
- long, très long à compiler : une modification d'une seule ligne oblige à recompiler tout le programme !
- Il vaut mieux éviter de mettre tous ses œufs dans le même panier (effacement malheureux d'un fichier)
- Et si on veut réutiliser une partie du programme ?

# Programmation modulaire

Programmer en utilisant des modules :

un module est un ensemble de fonctions (prototypes + définitions) que l'on met dans des fichiers à part : un programme sera donc un ensemble de fichiers qui seront réunis lors des différentes étapes de compilation et d'édition des liens.

Les environnements de développement savent réunir les fichiers (Visual C++, Dev-C++), il est possible de le faire 'à la main' (environnements Unix/Linux avec le compilateur gcc).

Plus facile à maintenir, possibilité de séparer les tâches lorsque l'on travaille en groupe, moins vulnérable, réutilisable.

# Programmation modulaire

Les fichiers utilisés pour générer un programme :

- un fichier `.c` ne contenant que le programme principal, souvent nommé `main.c` par habitude, mais rien n'y oblige, il peut tout à fait porter un nom différent, il faut juste **qu'il contienne la fonction nommée `main()`**.
- Des modules, qui sont des couples de fichiers `.c` et `.h`

`.c` : contient les définitions des fonctions (donc des instructions);

`.h` (h signifie header ou entête), contenant les prototypes des fonctions et des descriptions de constantes, de types, mais **AUCUNE** instruction.

Un fichier `.c` se compile, un fichier `.h` ne se compile pas.

# Contenu d'un fichier .h

Le fichier .h d'un module contient toutes les informations que d'autres modules ont besoin de connaître.

On y trouve donc :

- les prototypes des fonctions exportées (utilisées par d'autres modules)
- les définitions des types exportés
- les définitions des constantes exportées

Il en existe déjà un certain nombre : `stdio.h`, `stdlib.h`, `math.h`, etc...

# Contenu d'un fichier .c

Le fichier .c d'un module contient toutes les définitions des fonctions ainsi que les informations que les autres modules n'ont pas besoin de connaître.

On y trouve donc :

- les prototypes des fonctions dites locales (utilisées par ce module uniquement)
- les définitions des types locaux
- les définitions des constantes locales

ainsi que les définitions de toutes les fonctions : les fonctions locales ET les fonctions exportées.

# Inclusion des fichiers .h et directives

La question se pose toujours de savoir quand inclure un fichier .h grâce à la directive `#include`, et surtout où (dans quel fichier) inclure un fichier .h.

Il existe pour cela une règle très simple : on n'inclue un fichier .h que lorsque l'on en a besoin !

# Inclusion des fichiers .h et directives

Prenons l'exemple d'un programme réparti de la manière suivante :

un module `points`, définissant le type `t_p3d` (point dans un espace à 3 dimensions, ce sont ses coordonnées nommées  $x$ ,  $y$  et  $z$ ), et permettant d'utiliser des fonctions **`afficherP`** et **`saisieP`**.

Un module `transfos`, permettant de faire des calculs sur des points en 3D, en commençant par une fonction nommée **`symetrique`** calculant le point symétrique d'un point  $P$  par rapport à l'origine  $(0,0,0)$ .

Un programme principal qui fait la saisie d'un point  $P$ , l'affichage de ses coordonnées, le calcul du point  $Q$  symétrique de  $P$  et qui affiche les coordonnées de ce point  $Q$ .

# Le module point

Le module point est constitué des fichiers point.c et point.h

point.h contient :

la définition du (ou des types) pour représenter un point, il s'agira d'un type composé nommé **t\_p3d**.

les prototypes des fonctions **afficherP** et **saisirP**.

```
typedef struct s_p3d
{
    double x,y,z;
} t_p3d, *p_p3d;

extern void saisirPoint(p_p3d);
extern void affichePoint(t_p3d);
```

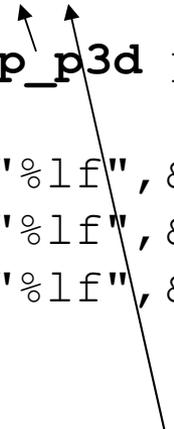
# Le module point

Le fichier `point.c` contient les définitions des fonctions. Puisque les fonctions utilisent les types `t_p3d` et `p_p3d` définis dans `point.h`, il faut inclure ce fichier.

```
#include "point.h"

void saisirP(p_p3d ptr_point)
{
    scanf("%lf", &(ptr_point->x));
    scanf("%lf", &(ptr_point->y));
    scanf("%lf", &(ptr_point->z));
}

void afficherP(t_p3d point)
{
    printf("[%6.3lf | %6.3lf | %6.3lf]\n", point.x, point.y,
point.z);
}
```



# Le module point

On veut par contre avoir une saisie sécurisée des valeurs entrées, en utilisant une fonction **saisieSec**, que l'on doit programmer. Cette fonction permettra de s'assurer que l'on a bien saisi une valeur à virgule, et pas un caractère ou du texte.

Cette fonction appartient au module point, mais n'a pas besoin d'être exportée, car seule la fonction **saisirP** va l'utiliser.

Il s'agira donc d'une fonction locale au module point, donc son prototype et sa définition se trouveront dans le fichier point.c. aucun autre module n'a besoin de connaître son existence.

Remarque : le fait de ne pas faire directement la saisie sécurisée dans saisirP est justifié, car on fait 3 fois le même contrôle (une fois pour chaque saisie), donc on peut utiliser une fonction.

# Le module point

```
// prototype de la fonction locale  
  
double saisieSec(void);  
  
// définition des fonctions locales  
  
double saisieSec(void)  
{  
    double val;  
  
    do  
    {  
        printf("Entrez une valeur à virgule :");  
        fflush(stdin);  
    }  
    while ( scanf("%lf",&val)!=1);  
  
    return val;  
}
```