# Compilation

Rappel de sémantique : le chemin vers un fichier est :

```
~/parent_dir/filename.ext
```

On croisera ici les extensions `o`, `h`, `c`, `out`, `ml`, `mli`, `cmi`, et `cmo`

*Remarque : l'extension ne modifie pas le fichier et n'est même pas "nécessaire", tout n'est que convention afin de donner des indications aux programmes et à l'utilisateur.*

## Compilation d'un seul fichier

### En C

```
gcc filename.c
```

### En OCaml

```
ocamlc filename.ml
```

```
ocamlopt filename.ml
```

Le fichier est produit sous le nom `a.out` et exécutable avec cette commande :

```
./a.out
```

On peut changer le nom de l'exécutable on ajoutant `-o exec_name`

## Compilation de plusieurs fichiers

### En C

On pourrait utiliser la commande suivante mais nous allons procéder autrement

```
gcc fileA.c fileB.c fileC.c
```

Nous allons compiler les fichiers un par un afin de rendre le projet plus modulaire

Pour exporter des fonctions on utilise une fichier d'en-tête `filename.h` dans lequel on y met les prototypes de no fonctions/variables

```
#ifndef A_H
#define A_H

// prototypes

#endif
```

On l'importe ensuite dans les fichiers `.c` qui en ont besoin en ajoutant la ligne`#include "filename.h"` avec les autres importations.

*Remarque : Par convention (et pour éviter des problèmes après modifications du code) on importe le fichier d'en-tête dans le fichier de code*

On peut compiler un fichier (dont on sera dépendant) avec la commande suivante

```
gcc -c filename.c
```

On obtiendra alors un fichier `.o`

Enfin on compile le projet avec cette commande après avoir compiler chaque fichier

```
gcc -o nom_exec a.o b.o c.o d.o
```

**En OCaml**

C'est assez similaire au C

Les fichiers `.ml` sont ceux dans lesquels nous codons tandis que les fichiers `.mli` contiendront les prototypes de ce que l'on exportera

Pour compiler individuellement on utilise la commande

```
ocamlc -c filename.ml
```

Elle produit deux fichiers

- `filename.cmo`
- `filename.cmi`

Grâce à ce fichier `.cmi` on peut compiler un fichier `b.ml` qui dépend de `a.ml` après avoir compiler ce dernier sans rien faire d'autre

*Remarque : c'est à peu près l'équivalent des* `.h` *produits automatiquement*

On peut tout compiler avec

```
ocamlc -c nom_exec fileA.cmo fileB.cmo
```

Nous allons cependant voir un procédé n'ayant pas besoin de compiler tous les fichiers et dans l'ordre (comme ce que nous avons fait en C)

On utilise alors un fichier `.mli` qui est vraiment l'analogue du `.h`

Exemple d'un fichier exportant une fonction f

`a.mli`

```
val f : int -> int
```

On peut ensuite exécuter un fichier `b.ml` s'appuyant sur `a.mli` avec cette commande

```
ocamlc -c a.mli b.ml
```

Désormais on utilisera cette commande si l'on souhaite compiler le fichier `a.ml`

```
ocamlc -c a.mli a.ml
```

## Remarques générales

- La grande différence avec le C est que le OCaml gère l'inférence de type, il peut produire lui-même son fichier d'en-tête `cmi`.
- Les fonctions qui n'ont pas pour vocation à être utilisées dans d'autres fichiers ne doivent pas être exportées.