Comilation séparée en C

À retenir

- La compilation se fait en deux étapes.
- La première nécessite que tout soit bien déclaré, et si ce n'est pas la cas on obtient des erreurs de type `implicit declaration of ...`.
- La deuxième nécessite que les fonctions appelées soient bien définies, et doit avoir accès à ces définitions, si ce n'est pas le cas on obtient des erreurs de type `undefined reference to ...`.
- La première étape produit un fichier compilé intermédiaire, qu'on nomme ….o.
- La deuxième étape produit un fichier exécutable, qu'on nomme sans extension. Pour cela, le point d'entrée est la fonction `main`, ainsi le code à compiler doit contenir une et une seule fonction `main`.
- un `Makefile` premet de définir des raccourcis pour lancer les compilations utiles.

En pratique (sans `Makefile`)

→ on déclare les fonctions dans un fichier `xxx.h` en donnant leur signature, par exemple `int nb_zeros(int* tab, int lg);`
→ on définit toutes les fonctions déclarées dans `xxx.h` dans un fichier `xxx.c` qui inclut `xxx.h` grâce à `#include "xxx.h"`
→ on compile ces définitions via la commande `gcc -c xxx.c -o xxx.o`.
→ on teste toutes ces fonctions dans la fonction `main` d'un fichier `test_xxx.c` qui inclut lui aussi `xxx.h` grâce à `#include "xxx.h"`
→ on compile ce programme de test avec les définitions de fonctions déjà compilées via la commande `gcc xxx.o test_xxx.c -o xxx`
→ on lance le programme de test avec `./xxx`

Pour éviter d'inclure plusieurs fois

Supposons qu'on ait déclaré dans `a.h` et codé dans `a.c` un objet A, en vue de travailler sur des objets B et D qui dépendent de A. On déclare l'objet B(resp. D) dans un fichier `b.h` (resp. `d.h`), qui contient `#include "a.h"` et on code les fonctions associées dans `b.c` (resp. `d.c`). On teste ces trois objets dans un même programme, codé dans le `main` du fichier `test.c`. qui contient `#include "b.h"` et `#include "d.h"`. Nul besoin d'importer `a.h`, car les déclarations de `a.h` sont incluses à travers `b.h`. Le problème c'est qu'elles sont incluses une deuxième fois à travers `c.h`, ce qui déclenche une erreur de redéfinition... Pour éviter ce problème, on conditionne la lecture du fichier au fait qu'une "variable" soit définie :

- à la première lecture du fichier `xxx.h`, la variable `XXX_H` n'est pas encore définie, alors on la définit puis le fichier est lu, donc les déclarations qu'il contient sont incluses ;
- à la seconde lecture du fichier `xxx.h`, la variable `XXX_H` est déjà définie, le fichier n'est pas relu et les déclarations qu'il contient ne sont pas re-incluses.

`a.h`
```
#ifndef A_H
#define A_h
...
#endif
```

`a.c`
```
#include "a.h"
...
```

`b.h` (idem pour `d.h`)
```
#ifndef B_H
#define B_h
..
#include "a.h"
...
#endif
```

`b.c` (idem pour `d.c`)
```
#include "b.h"
...
```

`main.c`
```
#include "b.h"
#include "d.h"
...
```