ASR TP1
Compilation de programmes C

Objectifs :

- Comprendre le processus de compilation de programmes C
- Organiser les fichiers d'un projet

1 Étapes de compilation

Reprendre l'exercice 1 de la feuille de TD 1 et compiler le programme en séparant les différentes étapes du processus de compilation (cf. cours). Observer les différents codes intermédiaires générés. Reprendre également les exemples du cours.

2 Headers

Un *header*, ou fichier en-tête, est utilisé pour séparer le prototype des fonctions de leurs implantations. Cela permet de faire de la compilation séparée : un fichier .c peut contenir un appel à une fonction présente dans un autre fichier juste en incluant le fichier .h correspondant. Lors de la phase de compilation, le compilateur vérifie que l'appel correspond bien au prototype et n'a donc pas besoin du code. On peut donc compiler tous les codes indépendamment pour obtenir les fichiers objets .o et si un .c est modifié, il n'y a pas besoin de recompiler tant que les prototypes de fonctions ne changent pas. Cela permet d'économiser énormément de temps de compilation sur de gros projets. Un autre avantage est que les fichiers .c peuvent être compilés séparemment donc en parallèle (plusieurs processeurs, plusieurs machines, fermes de compilation) pour accélérer encore le processus de compilation. Dernier avantage, il est possible d'utiliser des bibliothèques sans accès au code source si on dispose des fichiers d'en-têtes. Les fichiers objets obtenus sont combinés plus tard lors de la phase d'édition de liens pour former le programme exécutable.

Créer 2 fichiers : `fcts.c` et `fcts.h` contenant respectivement les implantations et les headers des fonctions `facto` et `fibo`. Le fichier `fcts.c` contiendra également la fonction `main` qui utilisera ces fonctions. Compiler en séparant les différentes étapes du processus de compilation.

3 Compilation séparée

1. Reprendre l'exercice précédent en déplaçant la fonction `main` dans un fichier `main.c`. Compiler toujours en séparant les différentes étapes.
2. Modifier soit le fichier `fcts.c` soit le fichier `main.c` (en ajoutant par exemple un appel à `printf`) et ne recompiler que ce fichier (en `.o`).
3. Refaire l'édition de lien et vérifier que votre modification fonctionne.

4 Compilation séparée... suite

1. Créer un fichier `simple.c` et son header associé contenant les fonctions `plusun` et `foisdeux`.
2. Compiler ce fichier.
3. Modifier le fichier `main.c` en ajoutant des appels à ces fonctions.

4. Recompiler le fichier `main.c`
5. Refaire l'édition de lien avec les fichiers `.o` obtenus.

5 Once only headers

Un fichier .h peut être inclus plusieurs fois dans un même fichier, par exemple si le fichier `main.c` inclus les fichiers `stdio.h` et `simple.h` et que le fichier `simple.h` inclus également le fichier `stdio.h` alors on pourrait se retrouver avec les définitions des fonctions d'entrées/sorties en double, ce qui ne pose pas de problème mais génère plus d'informations à traiter pour le compilateur. Pour éviter les inclusions multiples, on encadre le contenu des fichiers .h à l'aide d'une directive `#ifndef` du préprocesseur. Par exemple pour le fichier `simple.h` on peut encadrer le fichier de la manière suivante :

```
#ifndef SIMPLE_HEADER
#define SIMPLE_HEADER
// contenu du fichier...
#endif
```

Plus de détails ici : `https://gcc.gnu.org/onlinedocs/cpp/Once-Only-Headers.html`.

6 Erreurs de compilation fréquemments rencontrées

6.1 Undefined reference to symbol ...

Cette erreur se produit lorsqu'un fichier `.o` est oublié à l'édition de liens. Si vous n'avez pas rencontré cette erreur jusqu'ici vous pouvez la provoquer en reprenant l'exercice précédent et en enlevant par exemple le fichier `simple.o` lors de l'édition de lien. Il est important de savoir que dans ce cas, l'erreur vient soit :

- d'une erreur lors de l'appel d'une fonction : nom mal écrit,
- de l'absence du `.o` contenant la fonction lors de l'édition de liens.

6.2 Multiple definitions of symbol ...

Cette erreur se produit au moment de l'édition de lien lorsqu'un même symbole se retrouve dans plusieurs `.o` suite à l'inclusion d'un même `.h` plusieurs fois par exemple ou alors lorsque des symboles portent le même nom dans plusieurs fichiers (par exemple des fonctions différentes mais avec le même prototype).

7 Organisation d'un projet

Dans un projet, les sources, les fichiers d'en-têtes, la documentation et les fichiers générés sont généralement séparés afin de simplifier le développement suivant l'arborescence suivante :

- src : fichiers `.c`
- include : fichiers `.h`
- doc : documentation
- obj : fichiers `.o`
- bin : fichiers binaires exécutables générés

- lib : bibliothèques générées

Réorganiser vos fichiers suivant cette arborescence. Placer vous à la racine du projet et essayer de compiler votre programme à partir de cet emplacement.