C 9 : prototypes et compilation séparée

Juliusz Chroboczek

23 novembre 2023

1 Prototypes

Dans un programme constitué d'un seul fichier, une fonction peut être utilisée après le point où elle a été définie. Pour utiliser une fonction avant ce point, il faut la *déclarer* avant son utilisation à l'aide d'un *prototype*.

Le prototype est identique à l'entête de la fonction, mais est suivi d'un point-virgule « `;` ». Par exemple, la fonction définie par

```
int
min(int x, int y)
{
    if(x <= y)
        return x;
    else
        return y;
}
```

peut être déclarée à l'aide du prototype

```
int min(int x, int y);
```

2 Compilation séparée

Les programmes que nous avons écrits étaient entièrement contenus dans un seul fichier. Pour compiler un programme qui est contenu dans plusieurs fichiers, il suffit de les nommer tous sur la ligne de commande du compilateur C :

```
gcc -Wall main.c util.c
```

Déclarations et fichiers d'entêtes Pour utiliser une fonction dans un fichier autre que celui où elle a été définie, il faut la déclarer dans le fichier où elle est utilisée. On pourrait en principe dupliquer la déclaration dans tous les fichiers où une fonction est utilisée, mais ce serait laborieux et causerait des erreurs.

Au lieu de faire cela, on inclut les prototypes de toutes les fonctions dans un *fichier d'entête*, dont le nom se termine conventionnellement par « `.h` » (pour *header*). Ce fichier « `.h` » est ensuite inclus dans tous les fichiers « `.c` » qui ont besoin des déclarations à l'aide d'une directive `#include` qui est suivie du nom du fichier entouré de guillemets doubles :

```
#include "util.h"
```

Le d'entête est recherché dans le répertoire courant, et, s'il n'y se trouve pas, dans les répertoires du système.

Il est habituel d'inclure le fichier d'entête aussi dans le fichier qui définit la fonction, ce qui permet au compilateur de détecter les incohérences entre déclaration et définition.

Bibliothèques Une *bibliothèque* est un fichier contenant un ensemble de fonctions déjà compilées ; par exemple, la bibliothèque `libc` contient les fonctions `printf` et `scanf`, tandis que la bibliothèque `libm` contient `log` et `cos`.

Les fonctions contenues dans la bibliothèque sont déclarées dans un ensemble de fichiers d'entêtes. Pour s'en servir, on les inclut à l'aide de la directive `#include`, mais avec le nom du fichier entre chevrons :

```
#include <stdio.h>
```

À la différence de la syntaxe avec des guillemets, cette syntaxe fait que le fichier est recherché dans les répertoires du système mais pas dans le répertoire courant.

Lors de la compilation, il faudra spécifier la bibliothèque en ajoutant l'option `-l`*x* à la fin de la ligne de commande du compilateur, où *x* est le nom de la bibliothèque privé du préfixe `lib` :

```
gcc -Wall calcul.c -lm
```

La bibliothèque `libc` est utilisée automatiquement, il n'est donc jamais utile de spécifier explicitement `-lc`.

3 Compilation séparée

Lorsque tous les fichiers source sont énumérés sur la ligne de commande du compilateur, ils sont tous recompilés. Il est aussi posssible de ne recompiler que les fichiers qui ont changé depuis la dernière compilation.

Pour cela, on exécute une première fois le compilateur avec l'option `-c` sur chaque fichier pour le compiler séparément :

```
gcc -c -Wall main.c
gcc -c -Wall util.c
```

Cette phase de compilation crée, pour chaque fichier, un fichier dont le nom se termine par « `.o` » (« objet ») et qui contient le résultat de la compilation. Ces fichiers sont ensuite *liés* lors d'une nouvelle invocation du compilateur :

```
gcc main.o util.o
```

ou

```
gcc -o binary main.o util.o
```

Si un seul fichier source change, il suffit de recompiler ce fichier et de faire une nouvelle édition de liens ; les résultats de la compilation peuvent être réutilisés.

Makefiles La compilation séparée peut être automatisée à l'aide d'un outil nommé *make*. Pour utiliser *make*, il faut décrire toutes les dépendances dans un fichier nommé `Makefile`.

```
binary: main.o util.o
        gcc -o binary main.o util.o

main.o: main.c util.h

util.o: util.c util.h
```

Une invocation de la commande `make` effectue alors automatiquement toutes les phases nécessaires de la compilation.

Make est un outil archaïque, et la syntaxe des *makefiles* est fragile ; par exemple, une commande dans un *makefile* est forcément précédée d'un caractère tabulation. Il existe des outils plus modernes, mais je les trouve trop complexes et difficiles à utiliser.