Méthodes de Conception d'Algorithmes

C. TRABELSI & M. FRANÇOIS

TP9 -- Compilation séparée et fichier `makefile`

Compilation séparée et fichier `makefile`

Il s'agit ici de scinder le programme en plusieurs fichiers, contenant chacun une partie du code. Ceci permet une meilleure lisibilité et une meilleure compréhension du programme. La décomposition de vos fichiers peut se faire de la manière suivante :

- un fichier d'en-tête/header `librairies.h` regroupant toutes les librairies nécessaires pour votre programme ;
- un fichier d'en-tête/header `fonctions.h` regroupant les prototypes des fonctions ;
- un fichier `fonctions.c` contenant l'implémentation des fonctions dont les prototypes sont décrits dans le fichier `fonctions.h`;
- un fichier `main.c` contenant la fonction principale `main`, permettant de lancer le programme.

Pour compiler le projet, il faut d'abord compiler les fichiers ".c" via l'option "-c", pour générer les fichiers objets ".o". Une fois les fichiers ".o" créés, il faut les réunir dans une nouvelle compilation pour former l'exécutable final. Il est vrai que répéter à chaque fois cette procédure de compilation est contraignant, surtout si on a plusieurs fichiers. Le plus simple est de rassembler toutes ces commandes de compilation dans un fichier appelé `makefile`.

Un `makefile` est un fichier utilisé par le programme `make`, qui regroupe une série de commandes permettant d'exécuter un ensemble d'actions, typiquement la compilation d'un projet. Ce type de fichier est constitué essentiellement de plusieurs règles qui se suivent. La syntaxe d'une règle est la suivante :

Cible : dépendance1 dépendance2 ...
<——> commande1
<——> commande2
<——>

où :

- **Cible** désigne généralement le nom du fichier qui doit être généré par les commandes qui vont suivre ;
- **dépendances** représentent les liens requis pour la création de la cible, autrement dit les fichiers nécessaires permettant de créer la cible ;
- **commandes** constituent les lignes de commandes shell écrites par le programmeur, et qui seront exécutées au moment de la construction de la cible
- **<——>** constitue le caractère de tabulation.

Lors du parcours du fichier, le programme `make` évalue d'abord la première règle rencontrée (ici Cible), ou celle dont le nom est spécifié en argument (`make Règle`). L'évaluation d'une règle se

fait récursivement :

— les dépendances sont analysées : si une dépendance est la cible d'une autre règle, cette règle est à son tour évaluée ;
— lorsque l'ensemble des dépendances a été analysé, et si la cible est plus ancienne que les dépendances, les commandes correspondant à la règle sont exécutées.

Vous pouvez trouver un exemple de fichier `makefile` dans le cours 12.

EXERCICE

Pour cet exercice, vous devez utiliser la compilation séparée ainsi qu'un fichier `makefile`. Vous devez avoir quatre fichiers distincts :

- `"fonctions.c"` pour l'implémentation des fonctions,
- `"fonctions.h"` pour les prototypes des fonctions,
- `"main.c"` pour lancer le programme,
- `"makefile"` pour les commandes de compilation.

Ne pas oublier d'inclure le fichier `"fonctions.h"` dans les deux fichiers "c".

On dispose d'un véhicule permettant de faire 4 actions : avancer, reculer, tourner à droite et tourner à gauche. Quatre fonctions seront utilisées et dédiées à chacune de ces actions. On dispose d'un fichier de la forme suivante :

```
1 0 0 0
1 0 1 1
1 0 0 0
0 1 0 0
0 1 1 0
1 0 0 0
1 0 0 0
1 0 0 0
```

où chaque ligne contient 4 actions classées par ordre (avancer, reculer, tourner à droite, tourner à gauche). Si l'élément correspondant à une action est 1 (resp. 0) cela veux dire l'action sera (resp. ne sera pas) exécutée. Donc pour ce fichier la suite d'actions sera :
Avancer
Avancer
Tourner à droite
Tourner à gauche
...

► **1.** Écrire une fonction `AVANCER`, permettant de faire avancer le véhicule dans le cas où le paramètre d'entrée est égal à 1. Le prototype de la fonction est donné par :

```
void AVANCER (int action);
```

► **2.** Faire pareil pour les autres fonctions :

```
void RECULER (int action);
void TOURNER_D (int action);
void TOURNER_G (int action);
```

► **3.** Écrire une fonction `CONDUITE`, permettant de conduire le véhicule en faisant appel aux fonctions précédemment définies. Les ordres seront récupérés depuis un fichier en entrée.

Le prototype de la fonction est donné par :

```
void CONDUITE (FILE * fic);
```

Écrire la fonction `main()` associée à votre programme et testez-le.