Programmation Impérative 2 *L2 Informatique

Rappels - Compilation et Makefile

1 Prise en main

Les TP se font sous le systèmes Unix/Linux. Si vous ne l'avez pas encore fait, commencez par configurer votre espace personnel (alias, variables d'environnement, etc.). Puis créez un dossier approprié où vous mettrez les TP réalisés pendant les séances.

Pour écrire un programme C, il faut utiliser un éditeur. Vous êtes libre d'utiliser celui qui vous plaît le plus. Si vous n'en connaissez pas, nous vous proposons d'utiliser l'éditeur `emacs`. C'est l'un des éditeur les plus connus dans le monde Unix, il est très complet et s'adapte pratiquement à tous les langages en proposant des outils de coloration syntaxique, vérification de parenthésages, gestion de commentaires, compilation, etc. Il est également hautement configurable, cet aspect est toutefois en dehors du cadre de ce module et est laissé à l'appréciation de chacun.

La phase de compilation est effectuée à l'aide du programme `gcc` lancé avec la ligne de commande :

`gcc` [*options*] `fichier.c` [`-l`*librairies*]

Par défaut le programme créé s'appelle `a.out`, mais peut être modifié à l'aide de l'option `-o`. Les options importantes :

`-c` : supprime l'édition de liens ; produit un fichier objet (programme avec plusieurs fichiers).

`-E` : n'active que le préprocesseur (le résultat est envoyé sur la sortie standard).

`-g` : produit des informations symboliques nécessaires au débogueur.

`-Inom-de-repertoire` : spécifie le répertoire dans lequel doivent être recherchés les fichiers en-têtes à inclure (en plus du répertoire courant).

`-Lnom-de-repertoire` : spécifie le répertoire dans lequel doivent être recherchées les librairies précompilées (en plus du répertoire usuel).

`-o nom-de-fichier` : spécifie le nom du fichier produit. Par défaut, le fichier exécutable s'appelle `a.out`.

`-O, -O1, -O2, -O3` : options d'optimisations. Sans ces options, le but du compilateur est de minimiser le coût de la compilation. En rajoutant l'une de ces options, le compilateur tente de réduire la taille du code exécutable et le temps d'exécution. Les options correspondent à différents niveaux d'optimisation : `-O1` (similaire à `-O`) correspond à une faible optimisation, `-O3` à l'optimisation maximale.

`-S` : n'active que le préprocesseur et le compilateur ; produit un fichier assembleur.

`-v` : imprime la liste des commandes exécutées par les différentes étapes de la compilation.

`-W` : imprime des messages d'avertissement (warning) supplémentaires.

`-Wall` : imprime tous les messages d'avertissement.

N.B. : toutes les compilations devront systématiquement s'effectuer avec l'option `-Wall` (exemple `gcc -Wall -o monprog monfichier.c`). Si vous avez besoin d'utiliser des fonctions mathématiques, penser à inclure l'en-tête correspondant et à rajouter `-lm` à la fin de la ligne de compilation.

2 Programmation Modulaire et Makefile

La compilation d'un programme C constitué de plusieurs fichiers doit se faire en compilant chacun des fichiers pour obtenir un fichier objet `.o`, puis en effectuant une édition de lien. Pour faciliter cette phase de compilation, nous pouvons utiliser l'utilitaire `make` d'Unix. Cet utilitaire nécessite la définition d'un fichier `Makefile` - mis à la racine du répertoire contenant tous les fichiers sources par exemple - et qui permet de définir les différentes étapes de la compilation séparée en prenant en compte les dépendances entre les différents modules. Ultérieurement, si un module n'a pas été modifié, l'utilitaire le détectera et ne recommencera pas la compilation de ce module. Voici un exemple de fichier `Makefile` :

```
CC = gcc
CFLAGS = -Wall -O3
LDFLAGS = -lm #liens vers des libraires, comme pour la librairie math par exemple

LISTE_OBJ_PROG_MAIN= fichier1.o fichier2.o fichier3.o #liste des modules objets necessaires

main: $(LISTE_OBJ_PROG_MAIN)
    $(CC) $(CFLAGS) $(LISTE_OBJ_PROG_MAIN) -o mon_programme $(LDFLAGS)

fichier1.o: fichier1.c fichier1.h #rajouter des fichiers h si necessaire
fichier2.o: fichier2.c fichier2.h
fichier3.o: fichier3.c fichier3.h

clean:
    rm -f $(LISTE_OBJ_PROG_MAIN)
```