Université de Nice-Sophia Antipolis — Avril 2016
Polytech/Xidian — Durée: 1h30

Compilation séparée
Travaux Dirigés – Séance n. 11

---

exercice 1) On désire écrire un programme qui simule une calculatrice qui évalue des expressions en notation postfixée. Pour simplifier, les opérandes seront des entiers naturels, et les opérateurs seront les quatre opérations de base $+ - / \times$. Pour évaluer une expression postfixée, le programme a besoin d'une pile. L'expression est parcourue de gauche à droite. Chaque opérande est empilé et un opérateur trouve ses deux opérandes en sommet et sous-sommet de pile, qu'il remplace par le résultat de l'opération.

Exemples :

notation infixée	notation postfixée	résultat
$1 + 3$	$1\ 3\ +$	4
$2 + 3 \times 4$	$2\ 3\ 4\ \times\ +$	14
$2 \times 3 - 4 \times 5$	$2\ 3\ \times\ 4\ 5\ \times\ -$	–14

exercice 2) Le programme sera divisé en deux parties : implémentation de la pile et implémentation de la calculette.

La pile : écrivez dans deux fichiers, `pile.h` et `pile.c`, la définition de type de la pile et les fonctions de manipulation de la pile, `pilevide`, `empiler`, `dépiler`, `sommet` et `estvide`. Vous représenterez la pile à l'aide d'une structure chaînée.

La calculette : écrivez dans un fichier `cal.c`, le programme d'évaluation des expressions postfixées. L'expression est donnée par l'intermédiaire des paramètres de programme (*i.e.*`argc` et `argv`). Par ailleurs, la manipulation de la pile se fera *exclusivement* à l'aide des fonctions définies dans `pile.c`.

Compilation du programme : Pour obtenir l'exécutable, il faut faire la suite d'actions suivantes :

```
gcc -Wall -ansi -c pile.c
# génération de pile.o
gcc -Wall -ansi -c cal.c
# génération de cal.o
gcc -o cal cal.o pile.o
# édition de liens du tout et obtention de l'exécutable cal
```

ou compiler le tout en une seule fois, mais l'intérêt de la compilation séparée est de ne pas avoir à re-compiler un fichier si ce dernier n'a pas été modifié...

```
gcc -o cal cal.c pile.c
# compilation complète et obtention de l'exécutable cal
```

Make : en fait, pour compiler un programme réparti sur plusieurs fichiers, il sera plus simple d'utiliser la commande `make` avec le fichier `Makefile`. Le fichier `Makefile` décrit la façon de produire le programme exécutable à partir d'un graphe de dépendance des fichiers sources. Le programme

`make` s'occupera de (re-)compiler uniquement les fichiers nécessaires (ceux qui ont été modifiés) à la fabrication de l'exécutable. Pour le programme `cal`, le fichier `Makefile` aura la forme suivante :

```
CC = gcc              # le compilateur à utiliser
CFLAGS = -ansi -Wall # les options du compilateur
LDFLAGS =             # les options pour l'éditeur de liens
SRC = cal.c pile.c    # les fichiers sources
PROG = cal            # nom de l'exécutable
OBJS =  $(SRC:.c=.o) # les .o qui en découlent
.SUFFIXES: .c .o      # lien entre les suffixes

all: $(PROG)

# étapes de compilation et d'édition de liens
$(PROG): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $^      # $@ la cible  $^ toutes les dépendances

cal.o: pile.h
pile.o: pile.h
# le lien entre .o et .c
%.o: %.c
        $(CC) $(CFLAGS) -c $<          # $< dernière dépendance

# pour faire propre
clean:
        rm -f *.o *~ core $(PROG)
```

Enfin, pour lancer la compilation, il suffit d'exécuter la commande `make` dans un fenêtre terminale, ou mieux encore depuis `emacs`.