

I53 - Compilation et théorie des langages

- TD1 -

EXERCICE 1. Bootstrapping

On dispose d'un compilateur CompilA permettant de compiler un langage A.

- Décrire un cycle de développement d'un compilateur CompilB pour un langage B, écrit en langage A.

EXERCICE 2. Bytecode Python

Le module `dis` de Python permet d'afficher le bytecode Python. On considère le script Python suivant:

```
import dis

def f(x):
    ...

def calcul():
    return f(9)-(f(5)+f(2))

print(dis.dis(calcul))

>>>
   2      0 LOAD_GLOBAL              0  (f)
   2      2 LOAD_CONST               1  (9)
   4      4 CALL_FUNCTION            1
   6      6 LOAD_GLOBAL              0  (f)
   8      8 LOAD_CONST               2  (5)
  10     10 CALL_FUNCTION           1
  12     12 LOAD_GLOBAL              0  (f)
  14     14 LOAD_CONST               3  (2)
  16     16 CALL_FUNCTION           1
  18     18 BINARY_ADD
  20     20 BINARY_SUBTRACT
  22     22 RETURN_VALUE
```

Figure 1: Script Python

1. À la vue de la structure du bytecode, que peut on dire du fonctionnement de la machine virtuelle de Python chargée d'exécuter ce code ?

2. Quel est la signification des différentes instructions ?

EXERCICE 3. On considère le `makefile` suivant:

```
myprog: main.o matrix.o vecteur.o
gcc -Wall main.o matrix.o vecteur.o -o myprog

main.o: main.c
gcc -Wall -c main.c

matrix.o: matrix.c
gcc -Wall -c matrix.c

vecteur.o: vecteur.c
gcc -Wall -c vecteur.c
```

Figure 2: makefile

1. Quels fichiers seront recompilés si on modifie le fichier `main.c`, `matrix.c` ou `vecteur.c` ?
2. On rappelle que l'on peut définir des variables dans un *makefile* à l'aide de la syntaxe:

`nom_variable : une suite de caractères`

pour la définition et

`$(nom_variable)`

pour l'appel. Réécrire la règle de la cible `myprog` à l'aide de trois variables représentant respectivement la commande `gcc`, les différentes options de compilation et l'ensemble des fichiers objets.

3. Ajouter une cible *phony* permettant de supprimer l'ensemble des fichiers objets et fichiers de sauvegardes.
4. On rappelle également plusieurs variables prédéfinies:

- `$@`: nom de la cible de la règle;
- `$^`: ensemble des dépendances;
- `$<`: première dépendance;

Réécrire la règle de la cible `myprog` à l'aide des variables prédéfinies.

EXERCICE 4. (exam 2019)

On considère le `makefile` suivant. On suppose tous les fichiers déjà compilés et à jour.

```
mini.exe : scan.o expr.o symb.o
           gcc $^ -o mini.exe -lfl

scan.o : scan.c
        gcc -c scan.c

expr.o : expr.c
        gcc -c expr.c

symb.o : symb.c
        gcc -c symb.c

scan.c : scan.lex expr.h
        flex -o scan.c scan.lex

expr.h : expr.c

expr.c : expr.y
        bison -d -o expr.c expr.y
```

1. Quelles cibles seront recomplilées si l'on exécute la commande `make` après avoir modifié le fichier `symb.c` ?
2. Même question avec le fichier `scan.lex`.
3. Même question avec le fichier `expr.y`.