

TD 5 - Analyse syntaxique : grammaires attribuées et code MVàP

Installation du simulateur MVàP

Les sources du simulateur sont disponibles ici <https://richardg.users.greyc.fr/L3-Compil/sources-MVaP-3.2.tgz>.

Les instructions pour le faire fonctionner se trouvent dans le fichier `ALIRE.md`. On suivra les instructions et les tests pour s'assurer de bien comprendre son fonctionnement.

Rappel: Ce simulateur nécessite ANTLR.

On dispose également de deux outils pour les démos & tests.

1 Code MVàP

Soit le programme suivant en MVàP

qui, une fois assemblé, devient :

	Adr	Instruction
PUSHI 11	0	PUSHI 11
PUSHI 6	2	PUSHI 6
PUSHI 15	4	PUSHI 15
MUL	6	MUL
SUB	7	SUB
PUSHI 5	8	PUSHI 5
ADD	10	ADD
PUSHI 12	11	PUSHI 12
ADD	13	ADD
PUSHI 9	14	PUSHI 9
PUSHI 4	16	PUSHI 4
MUL	18	MUL
PUSHI 7	19	PUSHI 7
MUL	21	MUL
ADD	22	ADD
WRITE	23	WRITE
POP	24	POP
HALT	25	HALT

Qu 1. Commenter ce code. Que réalise-t-il ?

La MVàP, en mode debug, écrit à chaque pas :

- la valeur du compteur de programme (pc) ;
- l'instruction à cette adresse ;
- la valeur du « frame pointer » (toujours 0, quand il n'y a pas d'appel de fonction) ;
- le contenu de la pile ;
- la hauteur de la pile;

Le début de l'exécution donne la trace suivante :

pc		fp	pile	
0	PUSHI	11	0 [] 0	# On empile 11
2	PUSHI	6	0 [11] 1	# On empile 6
4	PUSHI	15	0 [11 6] 2	# On empile 15
6	MUL		0 [11 6 15] 3	
7	SUB		0 [11 90] 2	

Qu 2. Commenter ce début de trace. Compléter.

2 Production de code pour des expressions arithmétiques

Soit l'expression suivante : $8 * 6 - 12$

Qu 3. Quel arbre sera produit par l'analyse syntaxique de cette expression ?

Qu 4. Quel code MVàP doit être produit par le compilateur pour effectuer le calcul de cette expression ?

Qu 5. Compléter les règles de la grammaire suivante pour produire du code MVàP

```
expression returns [String code]
  : a=expression '*' b=expression
  {
    }
  | a=expression '+' b=expression
  {
    }
  | ENTIER
  {
    }
  ;
```

3 Variables globales

Soit le programme suivant en MVàP

et son code assembleur

	Adr Instruction
PUSHI 0	-----+-----
PUSHI 0	0 PUSHI 0
JUMP Main	2 PUSHI 0
LABEL Main	4 JUMP 6
PUSHI 7	6 PUSHI 7
STOREG 0	8 STOREG 0
PUSHI 2	10 PUSHI 2
STOREG 1	12 STOREG 1
PUSHI 1	14 PUSHI 1
PUSHG 0	16 PUSHG 0
PUSHG 1	18 PUSHG 1
MUL	20 MUL
ADD	21 ADD
STOREG 1	22 STOREG 1
PUSHG 1	24 PUSHG 1
WRITE	26 WRITE
POP	27 POP
HALT	28 HALT

Le début de l'exécution donne :

pc		fp	pile
=====			
0 PUSHI	0	0 []	0
2 PUSHI	0	0 [0]	1
4 JUMP	6	0 [0 0]	2
6 PUSHI	7	0 [0 0]	2
8 STOREG	0	0 [0 0 7]	3

Qu 6. Compléter et commenter ce début de trace.

4 Reconnaître des numéraux

Qu 7. Définir une grammaire attribuée qui reconnaisse les numéraux en anglais et calcule leur valeur. Ex : *forty two* → 42