TD 5 - Analyse syntaxique : grammaires attribuées et code MVàP

1 Code MVàP

Soit le programme suivant en MVàP

```
PUSHI 11
PUSHI 6
PUSHI 15
MUL
SUB
PUSHI 5
ADD
PUSHI 12
ADD
PUSHI 9
PUSHI 4
MUL
PUSHI 7
MUL
ADD
WRITE
POP
HALT
```

qui, une fois assemblé, devient :

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

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

2 Trace d'exécution

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.

3 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
        {

        }
    ;
```

4 Variables globales

Soit le programme suivant en MVàP

```
  PUSHI 0
  PUSHI 0
  JUMP Main
LABEL Main
  PUSHI 7
  STOREG 0
  PUSHI 2
  STOREG 1
  PUSHI 1
  PUSHG 0
  PUSHG 1
  MUL
  ADD
  STOREG 1
  PUSHG 1
  WRITE
  POP
  HALT
```

et son code assembleur

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

5 Reconnaître des numéraux

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