

# Compilation — TD 6

## Génération de code intermédiaire

J. Boender & G. Henry

2009–2010

**Exercice 1** Pour chacun des fragments de code suivants, construisez le code intermédiaire arborescent correspondant, mettez-le ensuite sous forme linéaire.

a)  $2 + (3 * 5) - 8$

b) `printint(2 + (3 * 5) - 8)`

On suppose maintenant que l'on se trouve au niveau (*level*) 2, que la fonction `f` est définie au niveau 3, que la variable `x` est définie au niveau 2 et à l'offset 2, que la variable `y` est définie au niveau 2 et à l'offset 3, que la variable `b` est définie au niveau 2 et à l'offset 4, et que la variable `z` au niveau 1 et à l'offset 2.

c) `x := 2 + 3 * f(5)`

d) `b := (x >= (y + 1))`

e) `2 * x + z`

f) `while(x < 3) do x := x - 1 done`

On suppose maintenant que l'on se trouve au niveau 4, que la variable `x` est au niveau 4 offset 2, et que la fonction `f` a été définie au niveau 2.

g) `x := f(3, 5)`

**Exercice 2** Construisez le code intermédiaire arborescent correspondant à aux fonctions suivantes :

a) `function f(x : int, y : int) = x * y + 1`

b) `function pair(x:int) = if x = 0 then 1 else impair(x-1)`  
`and impair(x:int) = if x = 0 then 0 else pair(x-1)`

**Exercice 3** On suppose maintenant qu'on se trouve au niveau 2, que la variable `a` est de type `arraint = array of int` et est définie au niveau courant, offset 2. Construisez le code intermédiaire correspondant au programme suivant :

```
a := arrint [10] of 0;
printint(a[5])
```

## Code intermédiaire arborescent

```
type binop =  
  | PLUS | MINUS | MUL | DIV  
  | AND | OR  
  | LSHIFT | RSHIFT | ARSHIFT | XOR  
type relop = EQ | NE | LT | GT | LE | GE  
type stm =  
  | LABEL of lbl  
  | JUMP of lbl  
  | CJUMP of relop × exp × exp × lbl × lbl  
  | MOVE of exp × exp  
  | EXP of exp  
and stms = stm list  
and exp =  
  | BINOP of binop × exp × exp  
  | MEM of exp  
  | WORDMUL of exp  
  | FP  
  | TEMP of Temp.temp  
  | ESEQ of stms × exp  
  | CONST of int  
  | STR of lbl  
  | CALL of lbl × exp list
```

## Code intermédiaire linéarisé

```
type exp =  
  | LBINOP of Ir.binop × exp × exp  
  | LMEM of exp  
  | LWORDMUL of exp  
  | LFP  
  | LTEMP of temp  
  | LCONST of int  
  | LSTR of lbl  
type stm =  
  | LLABEL of lbl  
  | LJUMP of lbl  
  | LCJUMP of Ir.relop × exp × exp × lbl × lbl  
  | LMOVETEMP of temp × exp  
  | LMOVEMEM of exp × exp  
  | LMOVETEMPCALL of temp × lbl × exp list  
  | LEXPCALL of lbl × exp list  
type stms = stm list  
type funbody = stms × exp
```

## Représentation des fonctions et des programmes

```
type  $\alpha$  proc_desc = {  
  entry : lbl; (* Point d'entrée de la fonction *)  
  numargs : int; (* Number of argument *)  
  numvars : int; (* Maximum number of escaping variable *)  
  code :  $\alpha$ ; (* Procedure's code. *)  
}  
  
type 'code prog = {  
  strings : (lbl  $\times$  string) list; (* Liste des chaînes de caractère *)  
  procs : 'code proc_desc list; (* Liste des fonctions *)  
}  
  
type ir = exp prog  
type lin = funbody prog
```