IFIPS – Cycle Ingénieur 2e année Informatique
Compilation 2006–2007

Travaux Dirigés n^{o}5
Génération de code

1 Procédures avec paramètres passés par valeur ou par référence

Soit le programme Pascal suivant :

```
program test;
var i,sum,facteur : integer;
procedure q (x:integer, y:integer);
  begin y := facteur * x + y end;
begin
  i := 1; sum := 0; facteur := 2;
  while i <= 10 do begin q(i+1, sum); i := i+1 end;
  writeln(sum);
end
```

1. Indiquer comment seront allouées les variables globales et locales dans la pile.
2. Donner une séquence d'instructions pour le code compilé de ce programme.
3. Lorsqu'un paramètre est passé par référence, il faut transmettre son adresse et non sa valeur. Reprendre les questions précédentes lorsque le paramètre `y` est passé par référence.

2 Récursion terminale

On se donne un mini-langage pour la manipulation d'expressions arithmétiques. Un programme de ce langage est composé de la déclaration de fonctions (déclarées par `fun`) ou de variables globales (déclarées par `val`). La grammaire est la suivante :

P	:= B E	E	:= `num`
B	:= ϵ	E	:= `id`
B	:= B D	E	:= `id`(F)
D	:= `val id` = E	E	:= `if` E `relop` E `then` E `else` E
D	:= `fun id` (L) = E	E	:= E `op` E
L	:= `id`	F	:= E
L	:= L, `id`	F	:= F, E

`op` (resp. `relop`) est un symbole terminal dont la valeur est l'un des opérateurs arithmétiques $\{+, -, *, /\}$ (resp. $\{<, \leq, =\}$). E représente une expression, D une déclaration de variable globale ou de fonction, L représente une suite d'identificateurs, F une suite d'expressions, B une suite de déclarations.

La valeur d'un programme formé d'une suite de déclarations B et d'une expression E est égale à la valeur de l'expression E évaluée dans l'environnement correspondant à B. Ce langage est compilé vers la machine à pile décrite en cours.
On appelle P le programme suivant :

```
val a = 4
fun square(z) = z*z
fun f(x,y)=square(x)+square(y)
val b = 2
f(a,b)+a
```

1. Donner pour le programme P une manière de stocker les variables globales et les paramètres des fonctions dans la pile. Indiquer pour chaque variable les codes d'accès en lecture ou en écriture.
2. Donner le code compilé correspondant au programme P.
3. On se propose de traduire chaque programme de notre langage vers le code de la machine à pile. On pourra supposer que toutes les déclarations de fonctions ont des noms distincts. On suppose également qu'une analyse a permis de relier chaque utilisation d'une variable à sa déclaration, ce qui nous permet de tester si la variable est locale ou globale et de connaître son décalage.

   La traduction est effectuée à l'aide de fonctions de codage $code_B$, $code_E$, $code_F$ ...pour les principaux non-terminaux de la grammaire. Ces fonctions sont spécifiées par des équations. Le début de la spécification est le suivant :

$$\begin{array}{ll}
code_P(B\ E) & = code_B(B)\ code_E(E) \\
code_B(\epsilon) & = [] \\
code_B(B\ D) & = code_B(B)\ code_D(D) \\
code_E(num) & = \texttt{PUSHI}\ num \\
code_E(E_1\ \texttt{op}\ E_2) & = code_E(E_1) code_E(E_2) code_{\texttt{op}}(\texttt{op}) \\
code_{\texttt{op}}(+) & = \texttt{ADD} \\
code_{\texttt{op}}(-) & = \texttt{SUB} \\
code_{\texttt{op}}(*) & = \texttt{MUL} \\
code_{\texttt{op}}(/) & = \texttt{DIV} \\
code_{\texttt{relop}}(<) & = \texttt{INF} \\
code_{\texttt{relop}}(\leq) & = \texttt{INFEQ} \\
code_{\texttt{relop}}(=) & = \texttt{EQUAL}
\end{array}$$

   Compléter cette spécification pour l'ensemble de la grammaire.

4. Si on suppose dans un premier temps que les fonctions définies ne sont pas récursives, que peut-on dire de la taille maximale de la pile utilisée lors de l'exécution de ce programme ?

   Si on suppose maintenant que les fonctions peuvent être récursives, que peut-on dire sur la taille de la pile ?

5. La récursion terminale, est un cas particulier de récursion où on peut limiter la taille de la pile. On dira qu'un identificateur f est *terminal* dans une expression e si l'une des conditions suivantes est vérifiée :
   - f n'apparaît pas dans e,
   - $e = f(e_1, \dots, e_n)$ et f n'apparaît dans aucun des e_i,
   - $e =$ `if` e_0 `then` e_1 `else` e_2 et f n'apparaît pas dans e_0 et est terminal dans e_1 et e_2.

   Une déclaration : `fun` $f(L) = E$ est récursive terminale si f est terminale dans E.
   On peut par exemple écrire une version récursive terminale de la factorielle :

   ```
   fun fact_it(n,m)=if n=0 them m else fact_it(n-1,n*m)
   fun fact(n)=fact_it(n,1)
   ```

   Décrire une manière de calculer si une déclaration de fonction est récursive terminale.

6. Soit f un identificateur terminal dans une expression e. Montrer que si le calcul de e fait un appel à $f(u_1, \dots, u_n)$ alors c'est que la valeur de e est la même que la valeur de $f(u_1, \dots, u_n)$.

7. Si le corps de la fonction récursive terminale $f(x_1, \dots, x_n)$ fait un appel à $f(u_1, \dots, u_n)$, on décide d'empiler les valeurs de $u_1, \dots, u_n$ à la place des arguments $x_1, \dots, x_n$ et de mettre la valeur de retour de $f(u_1, \dots, u_n)$ à la place de la valeur de retour $f(x_1, \dots, x_n)$. Comment faut-il modifier la génération de code correspondant à la production $E \coloneqq id(F)$ lorsque l'on compile le corps de la fonction récursive terminale f et que $id = f$?