TP2 - Langage VSL+

Le langage VSL (Very Simple Langage) a été introduit par J.P. Bennett dans le livre « Introduction to Compiling Techniques : a first course using ANSI C, LEX and YACC ». Le langage VSL+ est une évolution de VSL introduisant les tableaux et permettant une vérification de type complète.

Un programme est une suite de définitions de fonctions et prototypes de fonctions (voir plus bas pour le rôle des prototypes). La fonction main est la fonction principale du programme. Elle ne peut avoir d'arguments.

Tous les mots clefs de VSL+ sont écrits en majuscules et les identificateurs doivent être écrits en minuscules. VSL+ ne comporte que deux types de constantes : les entiers et les chaînes de caractères. Les chaînes de caractères ne sont utilisées que dans les ordres d'impression. Un entier est un entier naturel (donc non signé).

Les variables sont désignées par des identificateurs et doivent être déclarées. Elles peuvent être de type entier ou tableau d'entiers à une dimension. Les déclarations de variables sont de la forme :
`INT variable1 , variable2 , ... , variablen`
`variablei` est soit un identificateur, soit de la forme `ident[entier]`. Dans le premier cas il s'agit d'une variable de type entier et dans le second d'une variable de type tableau d'entiers dont la dimension est précisée entre les crochets.

Les expressions sont, d'une part, les expressions arithmétiques habituelles (opérateurs binaires +,-,* et /, et parenthésage) et, d'autre part, les appels des fonctions dont le type du résultat est INT (ce contrôle sera réalisé par les actions, pas par la grammaire) : `nomfonction(exp1 , ... , expn )` et les éléments de tableaux.

Avant toute utilisation une fonction doit être soit définie, soit prototypée (dans le cas d'une référence avant). La définition d'une fonction est de la forme :
`FUNC type nomfonction( var1 , ... , varn ) instruction`
type est le type du résultat et vaut INT ou VOID. La liste des arguments d'une fonction peut être vide. Afin de pouvoir être référencée par une autre fonction, une fonction peut être prototypée avant d'être définie. Un prototype de fonction a la forme :
`PROTO type nomfonction( var1 , ... , varn )`

Les instructions sont :

- les affectations : `partiegauche : = expression`, partiegauche étant une variable ou un élément de tableau.
- l'instruction de retour `RETURN` expression ; pour simplifier le TP, on n'interdira pas l'instruction `RETURN` dans une fonction VOID et on ne contrôlera pas (ni dans la grammaire, ni dans les actions) que l'exécution d'une fonction INT se termine toujours par un `RETURN`.

- une instruction d'impression : `PRINT` item$_1$, ... , item$_k$ où chaque item est soit une expression soit une chaîne de caractères : "chaînedechar". Les caractères autorisés sont les caractères ASCII imprimables et le caractère d'échappement `\n`.
- un ordre de lecture `READ` item$_1$, ... , item$_k$ ne pouvant lire que des entiers.
- deux conditionnelles : `IF expression THEN instruction FI`
  et `IF expression THEN instruction ELSE instruction FI`.
  Dans ces deux conditionnelles et dans l'instruction d'itération suivante, zéro joue le rôle de faux et toute autre valeur le rôle de vrai.
- une instruction d'itération : `WHILE expression DO instruction DONE`.
- un appel de fonction à résultat de type `VOID` (ce contrôle sera effectué par les actions, pas par la grammaire).
- enfin une instruction peut être un bloc de la forme :
  `{ listedeclarations listeinstructions}`
  où la liste des déclarations peut être vide mais pas la liste des instructions. La portée des déclarations suit la règle du plus petit bloc englobant. Les noms des fonctions sont de portée 0, les paramètres sont de portée 1 et les autres variables de portée ¿ 1.

Bien entendu, VSL+ supporte la récursivité. Voici un petit exemple de programme :

```
PROTO INT fact(k)
FUNC VOID main()
{
  INT i, t[11]
  i := 0
  WHILE 11 -i
  DO
  {
    t[i] := fact(i)
    i := i+1
  }
  DONE
  i := 0
  WHILE 11 -i
  DO
  {
    PRINT "f(", i, ") = ", t[i], "\n"
    i := i+1
  }
  DONE
}
FUNC INT fact(n)
  IF n THEN RETURN n* fact(n-1)
  ELSE RETURN 1 FI
```

Sur cet exemple, on voit que la fonction `fact` a été prototypée avant d'être appelée par la fonction `main`. La raison est sémantique : la fonction `main` fait référence, dans son texte, au nom `fact`, il faut donc que, lors de l'analyse de `main`, `fact` soit, au moins partiellement, connue. Toutefois, dans la phase d'analyse syntaxique (c'est-à-dire lors de la création de votre grammaire pour VSL+), aucune contrainte ne doit être imposée sur l'ordre des fonctions et des prototypes.