TP compilation : interprète VAR

Le langage VAR propose la possibilité de définir et d'appeler des fonctions, et offre à la fois des variables globales et des variables locales à une fonction. Avant de construire un compilateur de VAR vers IMP (semaine prochaine), nous allons écrire un interprète de ce langage.

1 Description du langage VAR

1.1 Spécification

1.1.1 Déclarations de variables

Le langage VAR permet de déclarer des variables avec le mot clé `var`, en leur associant ou non une valeur initiale. La déclaration prend l'une des formes suivantes

```
var_decl ::= var id ;
           | var id := n ;
```

où `id` désigne un nom et `n` une constante entière. Une variable déclarée mais non associée à une valeur sera automatiquement initialisée à la valeur `0`.

1.1.2 Expressions

Les expressions et expressions gauches de VAR sont identiques à celles d'IMP. Petite subtilité toutefois : une étiquette `id` peut maintenant désigner autre chose qu'une simple donnée globale.

1.1.3 Instructions

Les instructions `nop`, `exit`, `print`, `:=`, `if` et `while` conservent la même forme que dans IMP.

Les définitions d'étiquettes et les sauts `goto` disparaissent.

On a deux nouvelles formes d'instructions pour l'appel de fonction et le renvoi d'un résultat :

```
instr ::= ...
        | l_expr := l_expr ( expr , ... , expr ) ;
        | return ( expr ) ;
```

où *expr , ... , expr* désigne un nombre arbitraire d'expressions (potentiellement zéro) séparées par des virgules. La forme x `:=` $f(e_1, \ldots, e_n)$ désigne un appel à une fonction f avec les paramètres e_1 à e_n, dont le résultat est placé dans la variable x. La fonction f est une expression gauche, c'est-à-dire qu'on y accède par son adresse (en l'occurrence, l'adresse de son code).

1.1.4 Fonctions

La définition d'une fonction VAR de nom `f` prenant n paramètres formels x_1 à x_n a la forme

```
f(x_1, ..., x_n) {
  var_decl*
  instr*
}
```

où le corps de la fonction est constitué par une séquence de déclarations de variables locales, suivie d'une séquence d'instructions. La séquence d'instructions peut contenir une ou plusieurs instructions `return`.

1.1.5 Programmes

Un programme VAR est constitué d'une séquence de déclarations de variables globales, suivie d'une séquence de définitions de fonctions. On supposera que l'une des fonctions s'appelle `main` et vérifie les conditions suivantes :

— elle ne prend pas de paramètres,
— elle ne contient pas d'instruction `return`,
— elle se termine par une instruction `exit`.

1.2 Exemple

Voici un exemple de programme VAR.

```
var a := 97;

main() {
  var i := 0;
  while (i < 26) {
    print(a + i);
    i := succ(i);
  }
  print(10);
  exit;
}

succ(n) {
  return(n+1);
}
```

2 Objectifs du TP

On veut écrire un interprète pour le langage VAR, c'est-à-dire un programme qui prend en entrée un fichier source VAR (d'extension `.var`) et qui exécute ce programme.

2.1 Structure

L'interprète est constitué des fichiers suivants :

— `VAR.ml`, `FUNInstr.ml`, `IMPExpr.ml` et `Op.ml` définissent la syntaxe abstraite du langage VAR.
— `VARLexer.mll` et `VARParser.mly` génèrent un analyseur syntaxique traduisant un fichier source VAR en un arbre de syntaxe abstraite.
— `VAREvals.ml` définit la fonction d'interprétation des programmes VAR.
— `VARInterpreter.ml` est le fichier principal qui combine les éléments précédents.

2.2 Contrat de base

Compléter le fichier `VAREvals.ml` pour obtenir un interprète. Aucun autre fichier n'a besoin d'être modifié.

2.3 Extensions

Faire en sorte que la fonction `main` prennent des paramètres passés par la ligne de commande.