TP4 - Les visiteurs... sauce Scala

Fichiers de TP : `TP4_ACF.zip`

L'objectif de ce TP est de programmer en Scala un pretty printer et un évaluateur pour des programmes impératifs basiques. Le sujet est donc similaire au TD d'ACO qui vise le même objectif mais nous allons le traiter à l'aide de Scala pour illustrer certaines spécificités du langage. En particulier, l'utilisation des `case class` et des `match ... case` offre une alternative élégante à l'utilisation du patron de conception Visiteur.

1 Préambule

Pour utiliser Scala, si vous travaillez sur votre propre machine vous pouvez utiliser soit le bundle IntelliJ-Scala, soit Visual Studio Code (voir page du cours). Si vous travaillez sur les machines de l'ISTIC, nous vous conseillons d'utiliser Visual Studio Code : `/usr/bin/code`. Dans les deux cas, créez un repértoire ACF, puis :

1. Décompressez l'archive `TP4_ACF.zip` dans votre répertoire ACF.
2. Dans IntelliJ/VS Code, ouvrez le projet `TP4_ACF`.
3. Les sources se trouvent dans le répertoire `src/main/scala/tp4`
4. Pour créer d'autres objets/classes : Clic droit sur le répertoire `tp4` puis Nouveau>Scala class. Notez que contrairement à Java, un fichier Scala peut contenir plusieurs classes, objets, traits, ... Libre à vous d'organiser les fichiers comme bon vous semble !
5. Pour compiler/exécuter votre projet, tout passe par SBT (Simple Build Tool). Regardez les vidéos sur la page du cours.

2 Le type des arbres de syntaxes abstraits des programmes impératifs

Soit la grammaire suivante définissant des programmes impératifs basiques.

Grammaire	Exemple de programme

`Expression::= BinExpression | IntegerValue | Variable`
`BinExpression::= Operator; Expression; Expression`
`Operator::= "+" | "-" | "*" | "<" | "<=" | "=="`
`IntegerValue::= Int`
`Variable::= String`
 	
`Statement::= Assignment | Print | While | Seq | If | Read | Skip`
`Assignment::= Variable ":=" Expression`
`Print::= "print(" Expression ")"`
`While::= "while(" Expression ") do" Statement`
`Seq::= "{" Statement* "}"`
`If::= "if (" Expression ") then" Statement "else" Statement`
`Read::= "read(" String ")"`
`Skip::= "skip"`
	`{`
	`  x:= 0`
	`  y:= 1`
	`  read(z)`
	`  while ((x < z)) do`
	`  {`
	`  x:= (x + 1)`
	`  y:= (y * x)`
	`  print(x)`
	`  }`
	`  print(y)`
	`}`

On vous donne le code Scala définissant les classes nécessaires pour la représentation de ces programmes ainsi qu'un objet représentant une expression et un objet représentant le programme ci-dessus. Le code Scala définissant le type des expressions et les trois classes l'implémentant est le suivant :

```
sealed trait Expression
case class IntegerValue(i:Int) extends Expression
case class VariableRef(s:String) extends Expression
case class BinExpr(op:String, e1: Expression,e2: Expression) extends Expression
```

3 Un pretty printer

Voici un objet `PrettyPrinter` avec un extrait de l'opération `stringOf(e: Expression):String` permettant de produire la chaîne de caractère représentant l'expression `e`.

```
object PrettyPrinter{
  def stringOf(e:Expression):String={
    e match {
      case IntegerValue(i) => i.toString
      case VariableRef(v) => v
      ...
```

Définissez l'objet `PrettyPrinter` et équipez le d'une opération `stringOf(p: Statement):String` qui donne une chaîne de caractère représentant un programme. Tester votre fonction sur le programme `prog` donné. Il est conseillé d'utiliser la construction `match ... case`.

4 Un évaluateur

Définir un objet `Interpret` qui dispose d'une opération `eval(p:Statement,inList:List[Int]):List[Int]` permettant d'évaluer un programme et retourne la liste des entiers affichés successivement par les instructions `print` du programme. La liste `inList` contient, elle, la liste des entiers successivement saisis par l'utilisateur, *i.e.* lus par les instructions `read`. Une table de type `Map[String,Int]` vous sera nécessaire pour associer des valeurs entières à des noms de variables. **On considèrera qu'une variable non définie a comme valeur -1**. Il est également conseillé d'utiliser la construction `match ... case` pour la définition de `eval`.