TP 6 : Représentation intermédiaire

Dans ce TP, vous devez traduire votre langage en code intermédiaire. Du code vous est fourni (à adapter selon la définition de votre langage, imports à refactoriser, implantation des paires, etc...), vous indiquant l'architecture principale de la traduction en code intermédiaire, en suivant le cours à ce sujet.

L'essentiel des indications pour ce que vous devez réaliser dans ce TP se trouve dans le cours et dans le code (commenté). Quelques rappels :

Il y a deux étapes principales à cette construction :

- Définir la représentation intermédiaire à travers des classes Java. Ce code vous est fourni (mais vous pouvez être amené à le modifier ou l'étendre).[1]
- Traduire l'AST de votre langage dans cette représentation. (dans la classe `Result` vue en cours). La classe `translation/Translate` est à compléter.

Une difficulté est l'implémentation des cadres d'activation. Quelques `Frames` prédéfinis vous sont fournis en `translation/PredefinedFrames` à titre d'exemple (ils seront utiles pour traduire les lectures et écritures); ainsi qu'une bonne partie du code les concernants dans le package `translate`.

Remarques Ce TP est en grande partie un exercice de compréhension, sur un code qui reprend en les étendant les concepts abordés dans les travaux précédents. Il complète le cours sur la représentation intermédiaire à laquelle vous devez porter une attention soutenue.

Il s'agit ici de lire et comprendre le code fourni pour un langage avec appel de fonctions SDM (Software Data and Methods). Il vous faut compléter le fichier ir/translation/Translate.java

Quelques remarques :

- Lisez bien la grammaire du langage, ainsi que l'AST, notez bien les différences avec les langages précédents. En particulier la définition des programmes.
- Examinez également le package ir, avant de vous lancer dans la traduction
- Vous disposez de classes permettant d'afficher l'AST, les frames, les maps.

En particulier, quand vous aurez complété la classe Translate.java, et que vous aurez exécuté le main sur l'exemple fourni 'test.sdm' (n'hésitez pas à étendre votre jeu de tests), vous devez obtenir le résultat suivant :

[1] Les opérations relatives aux écritures et lectures dans la mémoire ne vous seront pas utiles si vous n'implémentez pas de tableaux ou de chaînes de caractères.

— Traduction en code intermédiaire —

```
Label L2
Frame{
  entryPoint=L0,
  exitPoint=L1,
  parameters=[reg0],
  result=reg1,
  locals=[],
  size=0
}
reg1 := (reg0 + 1)
goto L1
Frame{
  entryPoint=L2,
  exitPoint=L3,
  parameters=[],
  result=reg2,
  locals=[reg3, reg6, reg7],
  size=0
}
reg3 := 0
reg6 := call L0[reg3]
reg7 := call entryPrintInt[reg6]
reg2 := 0
goto L3
```