UE 4TIN603U – Compilation – Licence 3 – 2018-2019

Sujet du miniprojet

Les ressources pour faire le projet se trouvent à l'URL suivante: `https://www.labri.fr/perso/clement/enseignements/compilation/public/miniprojet.tar`

Modalités

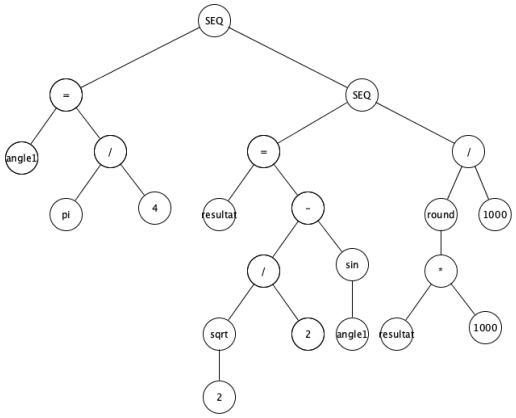
Le travail est
— À réaliser seul
— À rendre le mercredi 27 mars 2019 à minuit au plus tard.
— À envoyer par courriel à l'enseignant de son groupe de TD.
— Le sujet de l'*email* est `compilation <prénom> <nom> <numéro du groupe de TD>` et le fichier attaché est une archive `tar` portant le nom `<prénom>.<nom>.<numéro du groupe de TD>.tar`

Le non respect de ces modalités conduit l'étudiant.e à ne pas voir son projet noté ou corrigé (il.elle aurait alors zéro comme note pour ce travail).

Introduction

Ce projet est la réalisation d'un arbre de syntaxe abstraite pour un petit langage qui permet d'écrire une expression arithmétique contenant des variables. Après avoir réalisé l'arbre, il sera affiché puis évalué.

Par exemple $angle1 = pi/4; resultat = sqrt(2)/2 - sin(angle1); round(resultat * 1000)/1000$ donne l'arbre suivant :



et son évaluation donne 0.0

Mise en route

Nous fournissons les fichiers suivants :

```
|__ README
|__ build.xml
|__ data
|__ |__ input
|__ lib
|__ |__ beaver-ant.jar
|__ |__ beaver-cc.jar
|__ |__ beaver-rt-src.jar
|__ |__ beaver-rt.jar
|__ |__ jflex-full-1.7.0.jar
|__ parser
|__ |__ ParserExpr.grammar
|__ scanner
|__ |__ ScannerExpr.jflex
|__ src
    |__ Main.java
    |__ abstractTree
        |__ AbstTree.java
        |__ AbstTreeInt.java
        |__ Environment.java
        |__ EnvironmentInt.java
```

Les fichiers `ScannerExpr.jflex`, `ParserExpr.grammar`, et `Environment.java` doivent être complétés. Un ensemble de classes doivent être écrites dans `src/abstractTree`. Les commentaires doivent être écrits dans le code et éventuellement dans `README`. Enfin, le fichier `data/input` pourra être adapté au projet personnel.

Les autres fichiers ne doivent pas être modifiés.

La compilation se fait avec la commande `ant compile`, la construction de l'archive à envoyer à l'enseignant se fait grâce à la commande `ant dist`.

Analyse lexicale et analyse syntaxique

La grammaire pour le projet est la suivante :

```
Program =
        Declarations ';' Expression

Declarations =
        Declarations ';' Declaration
        | Declaration
```

```
Declaration =
        ID '=' Expression
        ;

Expression =
        Expression '+' Expression
        | Expression '-' Expression
        | Expression '*' Expression
        | Expression '/' Expression
        | '-' Expression
        | '(' Expression ')'
        | UFCT '(' Expression ')'
        | BFCT '(' Expression ',' Expression ')'
        | ID
        | INTEGER
        | FLOAT
        | 'pi'
        | 'e'
```

`UFCT` est le nom d'une fonction à un argument. `BFCT` est le nom d'une fonction à deux arguments, `ID` est un identificateur, `INTEGER` et `FLOAT` sont respectivement des entiers et des nombres à virgule flottante.

La sémantique du langage est la suivante :

— Une déclaration (`Declaration`) affecte la valeur de l'expression à la variable `ID`.

— Les expressions et les appels de fonctions s'évaluent de façon classique, la valeur d'une variable (`ID`) est la valeur précédemment enregistrée, `pi` est le nombre π et `e` est le nombre népérien.

Exercices

1. Choisir trois fonctions binaires et trois fonctions unaires de `java.lang.Math` et compléter le projet pour obtenir une analyse syntaxique complète en adaptant `data/input`.

   À ce moment du projet, on pourra temporairement commenter les quelques lignes de `main.java` qui attendent une sémantique qui n'a pas encore été implémentée.

   Le résultat doit être un message d'erreur en cas d'erreur dans l'`input`, rien sinon.

2. Implémenter la classe `Environment` qui permet de manipuler une table des symboles.

3. Implémenter des classes qui étendent la classe abstraite `AbstTree`. Ces différentes classes vont permettre de distinguer le type de noeud de l'arbre de syntaxe abstraite. À ce moment du projet, on pourra décommenter les quelques lignes de `main.java` qui attendent une sémantique, et visualiser l'arbre construit.

   Le résultat doit être l'affichage de l'arbre de syntaxe abstraite.

4. Implémenter la méthode virtuelle pure dans chacune de ces classes. Cette méthode va renseigner l'attribut `Double value` qui contient la valeur pour le noeud.

   Le résultat doit être l'affichage de la valeur correspondant à l'expression (`0.0` dans notre exemple).

Astuce : comment une déclaration ou une liste de déclarations pourrait avoir une valeur propre ? La réponse est la suivante : une affectation a comme évaluation la valeur affectée, une liste de déclarations a comme évaluation la dernière des évaluations.