POLYTECH LILLE

Université Lille1 Sciences et Technologies

Département IMA / 4A (S8)
Informatique Fondamentale 2011/2012

Laure Gonnord
http://laure.gonnord.org/pro/

Projet Tutoré

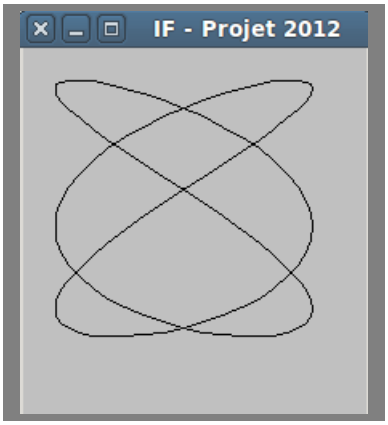
Réalisation d'un compilateur de dessins

Introduction

Il s'agit d'écrire un compilateur de langage de courbes en Java. Un dessin sera donné sous forme d'un fichier texte, et votre programme sera capable de l'interpréter et de l'afficher dans une fenêtre. Par exemple si le fichier `dessin.txt` contient une description de dessin :

```
./launch.sh dessin.txt
```

fournira l'affichage :



Ce sujet est largement inspiré de http://www.enseignement.polytechnique.fr/profs/informatique/Philippe.Chassignet/01-02/INF_431/dm_1/dm_1a.html, avec l'aimable autorisation des auteurs.

1 Plantons le décor !

Syntaxe concrète textuelle et sémantique des programmes Nous définissons maintenant le langage textuel complet sous forme de grammaire BNF :

```
PROG       ::= LISTINST quit

LISTINT    ::= INST LISTINT
             | INST

INST       ::= INST_EXPR  ;
```

```
              | INST_DEF   ;
              | INST_PRINT ;
              | INST_PLOT  ;

INST_EXPR  ::=  E

INST_DEF   ::=  set TKVAR = E

INST_PRINT ::=  print E,E:e2 for TKVAR = E .. E
              |  print E,E:e2 for TKVAR = E .. E by E

INST_PRINT ::=  plot E,E:e2 for TKVAR = E .. E
              |  plot E,E:e2 for TKVAR = E .. E by E

E          ::=    (expression arithmétique)
```

Explications :

- Un programme (`PROG`) est donc une suite d'instructions terminé du mot-clé `quit`.
- E est une expression arithmétique formée de nombres, constantes, identifiants, opérateurs arithmétiques (+, *, ...) et d'opérateurs fonctionnels (cos,sin,max). La grammaire de E sera à compléter dans la section 2.1.
- Une instruction (`INST`) est soit une `INST_EXPR` (évaluation d'une expression), soit `INST_DEF` (définition de variable), soit une `INST_PRINT` (impression d'expression), soit `INST_PLOT` (dessin d'un ensemble de points). Chaque instruction est terminée par un point virgule.
- Une instruction-expression est constitué uniquement d'une expression arithmétique (avec des variables).
- Une instruction-définition permet de donner une valeur à une variable.
- Une instruction-print permet d'imprimer une liste de couples d'expressions
- Une instruction-plot permettra d'afficher le dessin d'une courbe paramétrée.

L'évaluation d'un programme consiste à réaliser l'évaluation dans l'ordre de l'écriture des instructions :

- L'évaluation d'une instruction-expression consiste à imprimer la valeur de cette expression.
- L'évaluation d'une instruction-définition consiste à réaliser l'association entre la variable définie et une valeur d'expression.
- L'évaluation d'une instruction-print consiste à imprimer des couples d'expression en faisant varier les valeurs d'une variable d'une valeur `min` à une valeur `max` avec un incrément de 1 ou autre.
- L'évaluation d'une instruction-plot consiste à calculer les mêmes points que l'instruction-print correspondante, mais à les relier par des segments de droite dans une fenêtre graphique.

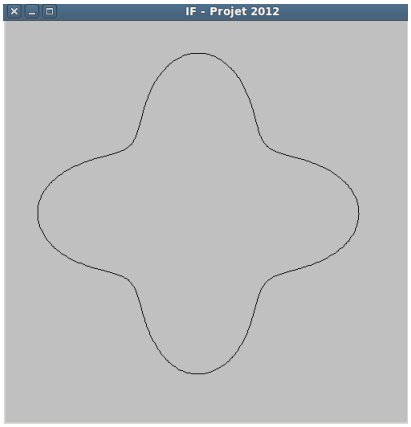
Exemple Un exemple de programme complet et (une partie de son) interprétation est décrit Figure 1. Vérifier que ce programme est bien un programme syntaxiquement correct et réaliser son interprétation "à la main". Vous fournirez cette explication pas-à-pas (il n'est pas demandé de calculer tous les points de la courbe) dans votre rapport.

```
plot 240+((4+cos(4*t))*cos(t)*40),240+((4+cos(4*t))*sin(t)*40) for t=0..2*Pi by 0.01 ;
set x=2;
set z=x+Pi+42;
print 100+t+2*z,100+2*t+42 for t = 0.. 3;
quit
```

(a) Définition

```
194.283185308  142.0
195.283185308  144.0
196.283185308  146.0
197.283185308  148.0
```

(b) Partie impression de l'interprétation



(c) Partie graphique de l'interprétation, un Blop

FIGURE 1 – Exemple de programme "blop" avec son interprétation

2 Travail demandé

2.1 Séance 1 - Analyse syntaxique

Préliminaires Tous les codes fournis pour le tutorat sont sur le SVN de l'école. On commencera donc par faire [1] :

```
svn checkout svn+ssh://synthe/ima4/ProjetIF
```

Cette archive a l'arborescence suivante :

```
api_graphique    // interface pour les dessins
expr             // expressions arithmétiques simples
launch.sh        // script pour lancer le programme
Makefile
testall.sh       // script pour lancer (tous) les tests
tests            // fichiers de tests
tools            // flex et cup
```

Vous devez tout d'abord récupérer les outils `jflex` et `jcup` permettant de compiler votre grammaire, ainsi que les fichiers `Makefile`, `launch.sh` et `testall.sh` déjà écrits. Enfin, on travaillera à partir des expressions arithmétiques vues en cours.

- Installer JFlex dans un répertoire de votre compte (copier, puis décompresser `jflex-1.4.3.tar.gz` qui se trouve dans le répertoire `tools`).
- Copier `java-cup-11a.jar` dans ce même répertoire d'installation.
- Faire un premier `checkout` de votre projet :
  `svn checkout svn+ssh://synthe/ima4/<nombinome>`

1. De l'extérieur, c'est plus compliqué, mais expliqué en cours de PA (voir la page WEB)

- Copier à la racine de ce répertoire <nombinome> `Makefile`, `launch.sh` (compilation, exécution de vos programmes), ainsi que `testall.sh`.
- Ajouter les droits d'exécution aux fichiers `launch.sh` et `testall.sh`.
- Créer (toujours dans <nombinome>) un répertoire `ifplot` qui va contenir les sources du projet. Y copier `expr.flex`, `expr.cup`, `ASTExpr.java` et `Analyseur.java`. Ajouter `package ifplot;` à chaque début des fichiers précédents.
- Éditer `Makefile` et `launch.sh` pour indiquer les bons chemins vers `jflex` et `java-cup`, puis vérifier que tout compile bien.
- Créer un répertoire `tests` et y copier l'ensemble des tests fournis (attention, pas de cp récursif à partir du répertoire fourni, sous peine de cassage de votre dépôt).
- Vérifier que `./launch tests/ex01.txt` ne fait pas d'erreur à l'exécution.
- `svn add`, `svn commit` :-)

REMARQUE 1 *JFlex a été téléchargé à l'adresse http://jflex.de/jflex-1.4.3.tar.gz et JavaCup à l'adresse http://www2.cs.tum.edu/projects/cup/java-cup-11a.jar.*

Expressions arithmétiques Vous avez donc récupéré dans le dépôt SVN la grammaire des expressions arithmétiques (suivie de `quit`) vue en cours (avec construction d'arbre syntaxique). Il s'agit maintenant de la modifier (dans le flex, le cup, suivant le cas) pour ajouter :

- les flottants (on gérera toutes les expressions et leur évaluation à l'aide de `Double`).
- les opérations moins et division binaire.
- les parenthèses.
- le moins unaire. On cherchera dans la documentation de JavaCup comment gérer ce moins.
- les fonctions cosinus (cos), sinus (sin).

On ajoutera aussi une fonction basique d'impression d'une `ASTExpr` dans le fichier `ASTExpr.java`.

Tester sur les programmes d'expressions fournis (dans `tests`, fichiers `ex01.txt` à `ex14.txt`). Un tgz fournissant une réponse à cette partie devra être envoyé par mail au plus tard **le dimanche 6 mai 18h** et fournira une base à la notation.

2.2 La suite

Étape 1 : Instruction Expression

- Écrire les classes Java (constructeur et méthode d'évaluation) `Prog.java` (classe pour un programme complet), `Instruction.java` (classe abstraite), et `InstructionExpr.java` (classe pour les instructions constituées d'une expression).
- Modifier la grammaire des expressions et les règles dérivées pour pouvoir parser les programmes constitués d'une liste d'expressions terminés par `quit`. Le résultat du parsing est une instance de la classe `Prog`, que l'on évalue *ensuite* à l'aide de l'appel à sa méthode d'évaluation, qui elle-même fait appel à la méthode d'évaluation de chaque classe `Instruction`.
- Tester (exemples fournis de `ex01.txt` à `ex15.txt`, éventuellement ajouter des exemples).

Étape 2 : Instruction Set Maintenant on va traiter les instructions de la forme :

```
set x = 7;
set y = x+8;
```

En ce qui concerne Flex/Cup, il faudra :

- Ajouter les token qui n'existent pas.
- Permettre les identifiants (et retourner un token de type `String`).

– Rajouter la règle de grammaire qui permet de reconnaître ces nouvelles instructions, et de construire une nouvelle classe `InstructionDef`.

Il faudra entre autres aussi étendre la classe des expressions de façon à ajouter les variables (attributs, méthodes).

Pour stocker les valeurs des variables on utilisera une hashtable que l'on augmentera au fur et à mesure de la lecture des instructions d'un programme (elle sera donc passée en paramètre des fonctions d'évaluation des instructions). Il faudra donc modifier les classes d'instructions écrites jusqu'à présent pour permettre ce passage par paramètre. **Il faut aussi que votre compilateur lance une exception lorsque l'on utilise la valeur d'une variable qui n'existe pas.**

Traiter cette nouvelle instruction et tester sur les exemples fournis.

Étape 3 : InstructionPrint Dans cette section on ajoute l'instruction "print/for" qui permet de réaliser des calculs itératifs de couples de valeurs.

```
set z = 1;
print 100+t+2*z,100+2*t+42 for t = 0.. 3;
quit
```

FIGURE 2 – Exemple ex20.txt

Sur cet exemple, l'effet de l'instruction est d'imprimer les couples de points $(102+t, 142+t)$ pour t valant successivement 0, 1, 2, et 3.

Il faudra donc :

– Modifier la classe `ASTExpr` en ajoutant une fonction de substitution d'une valeur à une variable, qui retourne une nouvelle instance de `ASTExpr`.
– Dans CUP rajouter les 2 règles de grammaire (print simple, print avec `by`) qui permettent de reconnaître ces nouvelles instructions, et de construire une nouvelle classe `InstructionPrint`.
– Dans la nouvelle classe `InstructionPrint` implanter la fonction d'évaluation.
– Tester bien correctement.

Sur l'exemple 2, le programme imprimera :

```
102.0 142.0
103.0 144.0
104.0 146.0
105.0 148.0
```

Votre compilateur doit lancer une exception lorsque l'on demande d'évaluer une expression qui n'est pas complètement calculable. (`for t=z..4` avec z non défini par exemple)

Étape 4 : InstructionPlot L'instruction-plot réalise les mêmes calculs que l'instruction-print correspondante, mais dessine des segments de droite qui rejoint les couples de points calculés.

```
set z = 1;
plot 100+t+2*z,100+2*t+42 for t = 0.. 3;
quit
```

FIGURE 3 – Exemple ex22.txt

Étapes :
- Dans CUP rajouter la règle de grammaire qui permet de reconnaître ces nouvelles instructions.
- Construire la classe `InstructionPlot` (ressemblant terriblement à la fonction `print`
- Dans un premier temps, la fonction d'évaluation imprimera sur le terminal les coordonnées des points des segments à dessiner.

Par exemple, sur l'exemple 3, le programme imprimera :

```
number of points : 4
draw line from (102.0,142.0) to (103.0,144.0)
draw line from (103.0,144.0) to (104.0,146.0)
draw line from (104.0,146.0) to (105.0,148.0)
```

Une fois les tests effectués, vous utiliserez l'API fournie (dans `api_graphique`) pour stocker les droites, puis les afficher dans une fenêtre graphique.

Étapes :
- D'abord tester l'utilisation de la bibliothèque fournie (il y a un README).
- Copier les fichiers `Graphique.java` et `GraphiqueInterface.java` dans le repertoire de votre projet
- Modifier la première ligne de ces fichiers pour être cohérent avec votre projet.
- Utiliser les fonctions fournies : déclarer un objet graphique dans `Prog` puis l'utiliser lors des appels à une `InstructionPlot` (la fonction d'évaluation a donc un paramètre de plus)
- Tester !

REMARQUE 2 *Deux instructions* `plot` *de suite pourront donner lieu à deux affichages successifs.*

Pour aller plus loin Regarder les extensions proposées par le site initial :
- Cadrage automatique (dans `Graphique.java`)
- Ajout d'axes.
- Choix des couleurs.

3 Consignes pour le rendu

La partie d'évaluation des expressions devra être rendue par mail (tgz –exclude-vcs) au plus tard le **le 6 mai 2012 à 18h**.

Vous devez rendre au plus tard **le 15 mai 2012 à 20h** (5 points par jour de retard) vos projets dans vos dépôts qui devront avoir la structure suivante :
- un fichier `Readme.txt` contiendra une description rapide de votre logiciel et de sa façon de l'utiliser.
- un Makefile et un `launch`.
- un répertoire `ifplot` contenant vos sources.
- un répertoire `tests` qui comprendra au moins les programmes de dessins fournis par l'énoncé
- éventuellement, un répertoire `old` contenant du code inutile.
- un fichier `nomdubinome.pdf` contiendra votre rapport. Le rapport ne comprendra pas plus de 5 pages, devra être clair et précis et notamment comporter les limitations de votre outil.

Attention ! votre dépôt SVN ne comprendra ni `jflex` ni `javacup` !