Perez Alexandre
Nasuti Mirco

29.12.2013

COMPILATEUR PNP

INTRODUCTION

Dans le cadre du cours de compilateur à la HE-Arc, nous avons réalisé un compilateur en Python. Ce compilateur permet de générer des fichiers SVG à partir de fichiers PNP qui respectent un langage que nous avons inventé pour l'occasion.

BUT

Notre projet a pour but de réaliser un compilateur simple en Python. Ce compilateur prend en entrée du code dans le langage inventé PNP vers le langage SVG (spécialisation de XML). Le langage PNP permet de décrire des éléments graphiques tels que des rectangles, des cercles ainsi que d'autres formes plus complexes comme des polygones. PNP doit également permettre de définir la couleur que prennent ces dessins. Le langage doit aussi fournir quelques structures de contrôle simples comme « if … then … », « while », etc.

LANGAGE PNP

DESCRIPTION

Le langage PNP est un langage que nous avons inventé spécialement pour ce projet. C'est ce langage qui est pris en entrée de notre compilateur. Nous l'avons créé en nous basant sur les fonctionnalités offertes par SVG mais en y modifiant la syntaxe et en y ajoutant des fonctionnalités supplémentaires telles que des structures de contrôle simples. Ce langage permet donc de facilement générer des graphismes répétitifs.

RÈGLES DE CODAGE

- Toutes les instructions doivent se terminer par un point-virgule « ; ».
- Tous les paramètres doivent être séparés par un double-point « : ».
- Il n'est pas obligatoire de préciser tous les paramètres.
- Les accolades « { » et « } » sont utilisés pour délimiter des blocs de code.
- Les variables booléennes peuvent prendre les valeurs « YES » ou « NO ».

OPÉRATEURS ARITHMÉTIQUES

Code	**Rôle**
+	Addition d'entiers
-	Soustraction d'entiers
*	Multiplication d'entiers
/	Division d'entiers
=	Affectation
%	Modulo

OPÉRATEURS LOGIQUES

Code	**Rôle**
<	Plus petit que
<=	Plus petit ou égal à
>	Plus grand que
>=	Plus grand ou égal à
==	Égal à

TYPES PRIMITIFS

Code	**Rôle**
Int	Entier
String	Chaîne de caractères
boolean	Booléen

ÉLÉMENTS DE DESSIN

Code	Paramètres	Rôle
Color	rgb(<int>, <int>, <int>) hex(<string>) name(<string>)	Couleur rgb -> couleur selon RGB hex -> couleur selon code hexadécimal name -> couleur selon nom anglophone (red, green, blue, black, white, yellow, orange, pink, …)
Point	x(<int>) y(<int>)	Point x -> abscisse y -> ordonnée
Line	p1(<point>) p2(<point>) fill_color(<color>) width(<int>)	Ligne p1 -> point de départ p2 -> point d'arrivé fill_color -> couleur width -> épaisseur
Circle	c(<point>) r(<int>) border_color(<color>) border_width(<int>) fill_color(<color>)	Cercle c -> centre r -> rayon border_color -> couleur du contour border_width -> épaisseur du contour fill_color -> couleur de remplissage
Rect	o(<point>) width(<int>) height(<int>) rx(<int>) ry(<int>) border_color(<color>) border_width(<int>) fill_color(<color>)	Rectangle o -> position width -> largeur height -> hauteur rx -> radius en abscisse ry -> radius en ordonnée border_color -> couleur du contour border_width -> épaisseur du contour fill_color -> couleur de remplissage
ellipse	c(<point>) ry(<int>) rx(<int>) border_color(<color>) border_width(<int>) fill_color(<color>)	Ellipse c -> centre rx -> radius en abscisse ry -> radius en ordonnée border_color -> couleur du contour border_width -> épaisseur du contour fill_color -> couleur de remplissage
customshape	p(<point>) border_color(<color>) border_width(<int>) fill_color(<color>)	Polygone particuliers p -> un sommet de la forme border_color -> couleur du contour border_width -> épaisseur du contour fill_color -> couleur de remplissage

Text	content(<string>) p(<point>) font(<string>) size(<int>) fill_color(<color>)	Texte content -> texte p -> position font -> style de police size -> taille de la police fill_color -> couleur

TRANSFORMATIONS

Code	**Paramètres**	**Rôle**
Rotate	angle(<int>) c(<point>)	Rotation angle -> angle de rotation c -> centre de rotation
Scale	sx(<int>) sy(<int>)	Étirement sx -> étirement en abscisse sy -> étirement en ordonnée
Translate	p(<point>)	Translation p -> vecteur de translation
Hide	h(<boolean>)	Masquage h -> activer/désactiver

STRUCTURES DE CONTROL

Code	**Rôle**

if <condition>
{
<body>
}
	Si <condition> est vraie alors <body> est exécuté.

while <condition>
{
<body>
}
	Tant que <condition> est vraie alors <body> est exécuté.

for <int> : <int>
{
<body>
}
	Pour les itérations d'une variable 'step' dans les bornes données par <int> : <int> alors <body> est exécuté.

apply <transformation>
{
<object>;
…
<object>;
}
	La transformation <transformation> est appliquée aux objets <object> listés entre les accolades.

MOT RÉSERVÉS

Dans le langage PNP, un certain nombre de mots ne peuvent être employés que dans des contextes particuliers. Voici la liste de ces mots réservés : color, point, line, circle, rect, ellipse, customshape, text, rotate, scale, translate, hide, if, while, for, step, apply, rgb, hex, name, x, y, p1, p2, fill_color, border_color, width, c, r, border_width, o, height, rx, ry, p, close, content, font, size, angle, sx, sy, h, int, string, boolean, YES, NO.

EXEMPLE DE CODE

Voici un « Hello world » en PNP :

```
display = hide : h(NO);
posText = point : x(20) : y(20);
helloText = text : content("Hello world") : p(posText);

apply display
{
  helloText;
}
```

LANGAGE SVG

DESCRIPTION

Le SVG est un format de données utilisé pour le dessin vectoriel. Dans ce format, les données sont stockées sous forme de code XML. Il est possible de lire facilement des fichiers de ce type dans un navigateur web. Notre compilateur a pour but de produire en sortie des fichiers SVG.

EXEMPLE

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <text x="20" y="20">Hello world</text>
</svg>
```

ANALYSE LEXICALE

Afin d'effectuer l'analyse lexicale, nous nous sommes aidés de PLY (Python Lex-Yacc). L'analyse lexicale se trouve dans le fichier « lex.py » dans lequel nous avons défini les mots réservés, les *tokens* et les *literals*.

ANALYSE SYNTAXIQUE

Afin d'effectuer l'analyse syntaxique, nous nous sommes aidés de PLY (Python Lex-Yacc). L'analyse syntaxique se trouve dans le fichier « parse.py » dans lequel nous avons défini la grammaire. Le code s'appuie sur le fichier fourni « AST.py ».

INTERPRÉTATION

Nous avons choisi une interprétation récursive car plus simple à mettre en place dans notre cas. Afin de faciliter la génération du SVG, nous avons utilisé la bibliothèque (python) svgWrite. L'utilisation d'une autre librairie au lieu de générer nous même du SVG directement nous a obligé nous adapter à cette librairie, ce qui fait que nous somme limité par celle-ci.

INSTALLATION

PRÉREQUIS

Ce compilateur nécessite d'avoir installé Python 3.x ainsi que Ply (Python Lex-Yacc), Graphviz et svgwrite[1].

EXTRACTION DES FICHIERS

Il suffit d'extraire les fichiers de l'archive fournie à l'endroit souhaité. L'installation est déjà terminée.

[1] https://pypi.python.org/pypi/svgwrite/

UTILISATION

LANCEMENT DU PROGRAMME

Il faut lancer un « invite de commande » (cmd.exe sur MS Windows ou un terminal comme xterm sur Linux). Il faut ensuite naviguer jusqu'à l'emplacement du programme (avec la commande « cd »).

La compilation peut se lancer avec la commande « python recInterpreter.py <fichier à compiler> ».

Il est également possible de lancer le programme simplement avec la commande « python recInterpreter.py ». Dans ce cas, un menu interactif vous guidera.

PROBLÈMES ÉVENTUELS

Si vous travaillez sur MS Windows et que la commande python n'est pas reconnue, vérifiez que Python soit ajouté à la variable d'environnement « PATH ».

Si vous êtes sous Linux et que des erreurs sont signalées au lancement, essayez de lancer le programme avec la commande « python » suivie du numéro de version de Python. Par exemple « python3.2 recInterpreter.py <fichier à compiler> ».

CONCLUSION

Nous avons obtenu un compilateur fonctionnel comme souhaité. Nous avons cependant pensé à quelques améliorations possibles pour une future version ; Telle que la possibilité de générer des animations, gérer la portée des variables, les nombres à virgules, et la gestion de fonction etc... Il pourrait également être intéressant d'ajouter les nombres complexes et des fonctions mathématiques comme sinus, cosinus ou racine. Ceci pourrait se faire assez facilement du fait que l'interprétation est faite en python et qu'il permet ces chose-là.