Projet
En Forme

Pour ce projet il vous est demandé de créer un langage de programmation permettant de dessiner des formes géométriques simples (et peut-être aussi des plus complexes). Il reprend les ressources pédagogiques [1, 2] proposées par Martin Ruckert de l'université des sciences appliquées de Munich.

Un programme dans ce langage pourrait ressembler à :

```
/* une primitive de base */
carré;
```

On pourrait aussi faire plus compliqué :

```
// variables globales
x = 200;
y = 200;

// une procédure
procédure kr(w,l) {
    [x,y] cercle;
    [x,y] rbv(1,1,0) secteur angle: w/2;
    pour i de w à 359 pas w
        [x,y] <i bleu secteur angle: w/2;

    [150+x,y-100] %200 rectangle;
    [200+x,y-50] {
        pour i de 1 à l pas 1 faire
            %100/i <i*-20 [0,50] rbv(1,i*1/l,0) rectangle;
    }
}

// appels de procédure
kr(45,10);
%20 kr(90,3);

// procédure récursive
procédure q(n,i) {
    si n>0 alors {
        rbv(i,1,i*0.5) rectangle;
        si i=0 alors j=1; sinon j=0;
        %50 {
            [0,100] q(n-1,i);
            [100,100] q(n-1,j);
            [100,0] q(n-1,i);
        }
    }
}

// appel de la procédure récursive
[100,400] %200 q(10,1);
```

Ces exemples donnent une idée de ce que doit pouvoir exprimer le langage :
- primitives (`rectangle`, `cercle`, `secteur`) ;
- coordonnées (`[10,200]`) ;
- échelles (20 % : `%20`) ;
- rotations (`<90`) ;
- couleurs (`bleu`, `rbv(...)`) ;
- tâches composites (`{}`) ;
- variables, procédures et structures de contrôle ;
- commentaires.

Les détails ne sont pas fixés. Vous pouvez formuler votre langage comme vous l'entendez, le principal étant que les programmes soient lisibles (pas de langage à la Brainfuck, Ook ou Spoon).

La spécification du langage fait partie de votre travail. Vous pouvez partir de zéro ou utiliser les exemples donnés comme point de départ. Attention tout de même : en partant de zéro vous rencontrerez peut-être des difficultés inattendues. Il est donc sans doute judicieux de commencer en vous basant sur les exemples, puis incorporer vos idées à travers différentes expérimentations pour, peut-être, obtenir un résultat final très différent des exemples.

Découpage du projet

Partie 1

Analyser les programmes donnés en exemple et identifier les différents tokens. Écrire un analyseur lexical avec lex/flex.

Partie 2

Analyser les programmes donnés en exemple et identifier les différentes structures de contrôle. Écrire une grammaire qui reconnaît le langage avec yacc/bison.

Partie 3

À quoi bon un super langage si personne ne le comprend ? Nous aurons donc besoin d'un traducteur de programmes écrits dans votre langage dans un langage compris par la machine. Généralement les programmes sont traduits en langage machine. Dans notre cas, un langage machine traditionnel n'est pas vraiment adapté : il serait plutôt intéressant de pouvoir visualiser le résultat des programmes. Nous choisissons donc de traduire les programmes en PostScript. Toute visionneuse PostScript ou imprimante compatible PostScript pourra ensuite être utilisée pour exécuter le programme traduit.

Les exemples donnés sont accompagnés d'images dont vous pouvez voir le code PostScript dans n'importe quel éditeur de texte (ce code n'est pas forcément optimal ou le plus esthétique).

La troisième tâche est donc de générer du code PostScript [3, 4] à partir de vos programmes.

Réflexion

Différentes questions à vous poser :

- En plus des entiers et des couleurs dans les exemples, faut-ils d'autres types de données ? Les variables sont-elles déclarées explicitement/implicitement ? Sont-elles typées ?
- Quelles opérations arithmétiques sont utilisées ? Autorise-t-on les opérateurs arithmétiques sur les couleurs ? Quel sens peuvent avoir `rouge + vert`, `blanc - bleu` ou `0.5*rouge` ?
- Comment sont séparées ou regroupées les instructions (à la C, à la Pascal, à la Python, un mélange) ?
- Comment gérer les erreurs de syntaxe ?

Consignes et évaluation

Ce projet est à faire en binôme et à rendre pour le vendredi 29 mars 23h55. Vous utiliserez le langage C pour développer votre projet. Il sera accompagné d'un compte-rendu d'au moins quatre pages qui présentera de façon claire et concise :

- la grammaire de votre langage ;
- comment vous avez réglé d'éventuels conflits (shift-reduce, reduce-reduce, etc) ;
- les tests effectués ;
- les difficultés rencontrées ;

Votre code source devra être accompagné d'un `makefile` et d'un ou plusieurs scripts permettant d'exécuter automatiquement les tests. Vous joindrez également un fichier README qui contiendra les commandes à utiliser.

Une mini introduction à PostScript

Les programmes PostScript sont de simples fichiers texte qui sont destinés à être interprétés par une visionneuse ou une imprimante PostScript. Une des particularités de PostScript est l'utilisation de la notation polonaise inversée pour définir les instructions. En effet, l'interpréteur PostScript est une machine à pile. Par ailleurs PostScript propose un mécanisme de dictionnaire intéressant.

Le concept de base d'une image PostScript est le chemin. Un chemin est créé à l'aide de l'opérateur `newpath` est utilisé par l'opérateur `stroke` ou `fill` afin de créer le chemin lui-même ou la forme pleine délimitée par le chemin. Parmi les opérateurs les plus importants, on retrouve :

- `moveto`, aller à la position indiquée ;
- `lineto`, dessiner une ligne jusqu'à la position indiquée ;
- `arc`, dessiner un arc ;
- `bezier`, dessiner une courbe de bezier.

Par exemple la primitive `carré` peut être générée de la façon suivante :

```
newpath
0 0 moveto
100 0 lineto
100 100 lineto
0 100 lineto
0 0 lineto
fill
```

Les coordonnées peuvent être manipulées grâce aux opérateurs `transform`, `rotate` and `scale`.

Références

[1] Martin Ruckert. Teaching compiler construction and language design : Making the case for unusual compiler projects with postscript as the target language. *SIGCSE Bull.*, 39(1) :435–439, March 2007.

[2] `http://w3-o.cs.hm.edu/~ruckert/compiler/klx`.

[3] *PostScript Language Reference.* Adobe Systems Incorporated, 3rd edition, 1999. `http://www.adobe.com/products/postscript/pdfs/PLRM.pdf`.

[4] *PostScript Language Document Structuring Conventions Specification.* Adobe Systems Incorporated, 1992. `http://partners.adobe.com/public/developer/en/ps/5001.DSC\_Spec.pdf`.