Projet compilation

Libert Robin
Zielinski Pierre
BA3 Info

14 mai 2018

Introduction

Dans le cadre du cours de compilation, dispensé par Véronique Bruyère pour la théorie et Alexandre Decan pour les TPs, nous avons créé un programme qui permet de générer facilement un fichier contenant du texte. L'utilisateur créera un fichier contenant des données, et le programme se chargera de créer un fichier texte à l'aide de ces données et grâce à un template dans lequel le programme va insérer les données.

Utilisation du programme

Lancer le programme avec la commande :
python dumbo_synth.py path/données path/template path/sortie
Si le fichier de sortie n'existe pas, le programme le créera.

Description des lexèmes

INITIAL STATE

BLOC_BEGIN ← {{

TXT ← `[\w|;|&|<|>|"|_|-|.|\|/|\n|\p|:|,| \t]+`

IN_STRING_STATE

APO ← '

STRING ← `[\w|;|&|<|>|"|_|-|.|\|/|\n|\p|:|,| ]+`

IN_CODE_STATE

APO ← '

BLOC_END ← }}

FOR ← for

IN ← in

DO ← do

ENDFOR ← endfor

IF ← if

ELSE ← else

ENDIF ← endif

PRINT ← print

INTEGER ← `\d+`

ADD_OP ← +|-

MUL_OP ← *|/

PAR_FERM ←)

PAR_OUVR ← (

DOT_COMMA ← ;

DOT ← .

ASSIGNEMENT ← :=

BOOLEAN ← true|false

VAR ← [\w]+

OPERATOR ← <|>|=|!=

BINOPERATOR ← or|and

Description de la grammaire

programme ← TXT | TXT programme

programme ← dumboBloc | dumboBloc programme

dumboBloc ← BLOC_BEGIN expressionList BLOC_END |BLOC_BEGIN BLOC_END

expressionList ← expression DOT_COMMA | expression DOT_COMMA expressionList

expression ← variableN ASSIGNEMENT globalExpression | variableN ASSIGNEMENT list

expression ← PRINT globalExpression

expression ← FOR variableN IN list DO expressionList ENDFOR | FOR variableN IN variable DO expressionList ENDFOR

expression ← IF globalExpression DO expressionList ENDIF | IF globalExpression DO expressionList ELSE expressionList ENDIF

globalExpression ← integerVar | string | variable | booleanVar| globalExpression BINOPERATOR globalExpression | globalExpression OPERATOR globalExpression | globalExpression ADD_OP globalExpression | globalExpression MUL_OP globalExpression | globalExpression DOT globalExpression

list ← PAR_OUVR stringListInterior PAR_FERM | PAR_OUVR integerListInterior PAR_FERM

stringListInterior ← string | string COMMA stringListInterior

integerListInterior ← integerVar | integerVar COMMA integerListInterior

variable ← VAR

variableN ← VAR

string ← APO STRING APO

integerVar ← INTEGER

booleanVar ← BOOLEAN

Gestion du if et du for

Pour chaque expression définie dans notre programme lors de l'analyse syntaxique, nous créons une classe correspondant à cette expression. Chaque classe aura une fonction d'évaluation que l'on appellera durant la phase d'analyse syntaxique et qui fera le travail nécessaire pour retourner la valeur correspondant à l'expression.

Une boucle for est une expression : expression ← FOR variableN IN list DO expressionList ENDFOR | FOR variableN IN variable DO expressionList ENDFOR
Nous faisons la distinction entre 2 types de variables. La première, variableN est un nom de variable dans lequel nous allons attribuer une valeur. La seconde, variable, est une variable déjà existante dans laquelle nous allons récupérer une valeur. Une boucle for est gérée par la classe ForExpression qui prend variableN en premier paramètre, list ou variable en second paramètre et une expressionList en dernier paramètre. A chaque itération, nous changeons la valeur à l'emplacement variableN dans un dictionnaire et l'envoyons à expressionList pour être évalué.

Notre IF est une expression : expression ← IF boolean DO expressionList ENDIF | IF boolean DO expressionList ELSE expressionList ENDIF
Un IF est géré par la classe IfExpression qui prend en premier paramètre un boolean en second paramètre une expressionList et en dernier paramètre, soit None soit une autre expressionList. Et tous ces paramètres seront géré par la focntion evaluate de IfExpression qui va à son tour appeler la classe evaluate des expressionList et du boolean et ainsi de suite.

Difficultés rencontrées

Lors de la réalisation, nous avons rencontré quelques soucis. Premièrement, nous avons eu des problèmes de version avec python et ply. Nous avons voulu utiliser des codes que l'on avait reçus en TP, mais ceux-ci ne fonctionnaient pas avec la version que nous avions de python et de ply. Nous nous sommes donc renseigné sur les changements de python 2 à python 3 et le problème fut résolu.

Ensuite, dans la boucle for, la partie expression avait besoin de savoir quel était la valeur mise à la variable. Ce problème a été résolu en mettant un dictionnaire en paramètre à chaque méthode «evaluate» en attribuant à chaque passage dans le for la bonne valeur à la variable du for.

Les integers, les strings, et les booleans étaient définis séparément dans l'analyseur syntaxique et cela créait un conflit reduce reduce car les trois éléments pouvaient être une variable. La solution a été de tout regrouper en un

seul type nommé globalExpression.

Les assignations de variables se faisaient dans la partie syntaxique. Cela créait un problème quand on analysait un fichier template car s'il y avait des assignations dedans celles-ci n'étaient pas faites. Le problème a été résolu en faisant l'assignation dans la partie sémantique.

Pour finir, l'assignation de variables contenant cette variable elle-même (exemple : i = i + 1) créait une récursion infinie lors de l'analyse sémantique. Le problème a été résolu en évaluant l'assignation avant de l'assigner (dans l'exemple on évalue le i + 1 avec la valeur actuelle de i dans le dictionnaire avant de mettre la valeur dans la variable i).