Conception d'un interpréteur
Projet de compilation

Dubuc Xavier & Vasseur Cyrielle

28 novembre 2018

Table des matières

1 Introduction

Il nous a été demandé de concevoir un interpréteur pour un langage de programmation de grammaire donnée (cf. Grammaire de base). Les outils utilisés pour la mise en oeuvre de ce projet sont les suivants :

— le langage de programmation **JFlex** utile à l'analyse lexicale,
— le langage de programmation **JCup** utile à l'analyse syntaxique et sémantique.

Des extensions à cette grammmaire sont demandées telles que la gestion des opérations de multiplication et de division, la lecture en console, la gestion des commentaires, ...

2 Grammaire de base

Nous allons tout d'abord rappeler la grammaire de base qui, comme nous le verrons dans la section suivante, a été légèrement modifiée.

<*programme*> → **debut** <*liste_instructions*> **fin**
<*liste_instructions*> → <*instruction*> <*fin_liste_instruction*>
<*fin_liste_instruction*> → <*instruction*> <*fin_liste_instruction*>
<*fin_liste_instruction*> → ϵ
<*instruction*> → **ID :=** <*expression*> ;
<*instruction*> → **lire (**<*liste_id*>**) ;**
<*instruction*> → **ecrire (**<*liste_expressions*>**) ;**
<*liste_id*> → **ID** <*fin_liste_id*>
<*fin_liste_id*> → **,** **ID** <*fin_liste_id*>
<*fin_liste_id*> → ϵ
<*liste_expressions*> → <*expression*> <*fin_liste_expression*>
<*fin_liste_expression*> → <*expression*> <*fin_liste_expression*>
<*fin_liste_expression*> → **,** <*expression*> <*fin_liste_expression*>
<*fin_liste_expression*> → ϵ
<*expression*>→ <*expression_simple*> <*fin_expression_simple*>
<*fin_expression_simple*> → <*operateur*> <*expression_simple*> <*fin_expression_simple*>
<*fin_expression_simple*> → ϵ
<*expression_simple*> → **(**<*expression*>**)**
<*expression_simple*> → **ID**
<*expression_simple*> → **INT**
<*operateur*> → **+**
<*operateur*> → **-**

3 Grammaire étendue

<*programme*> → **debut** <*liste_instructions*> **fin**
<*liste_instructions*> → <*instruction*> <*fin_liste_instruction*>
<*fin_liste_instruction*> → <*instruction*> <*fin_liste_instruction*>
<*fin_liste_instruction*> → ϵ
<*instruction*> → **ID** **:=** <*expression*> **;**
<*instruction*> → **lire** **(**<*liste_id*>**)** **;**
<*instruction*> → **ecrire** **(**<*liste_expressions*>**)** **;**
<*instruction*> → *if_block*
<*liste_id*> → **ID** <*fin_liste_id*>
<*fin_liste_id*> → **,** **ID** <*fin_liste_id*>
<*fin_liste_id*> → ϵ
<*if_block*> → **if** <*expression_booleenne*> **then** **{**<*liste_instructions*>**}** <*fin_if*>
<*fin_if*> → **else** **{**<*liste_instructions*>**}**
<*fin_if*> → ϵ
<*liste_expressions*> → <*expression*> <*fin_liste_expression*>
<*liste_expressions*> → <*expression_booleenne*> <*fin_liste_expression*>
<*fin_liste_expression*> → **,** <*expression*> <*fin_liste_expression*>
<*fin_liste_expression*> → **,** <*expression_booleenne*> <*fin_liste_expression*>
<*fin_liste_expression*> → ϵ
<*expression*> → <*expression*> **+** <*expression*>
<*expression*> → <*expression*> **-** <*expression*>
<*expression*> → <*expression*> ***** <*expression*>
<*expression*> → <*expression*> **/** <*expression*>
<*expression*> → <*expression_simple*> <*expression_simple*>
<*expression*> → <*expression_simple*>
<*expression_simple*> → **(**<*expression*>**)**
<*expression_simple*> → **ID**
<*expression_simple*> → **INT**
<*expression_simple*> → **REAL**
<*expression_simple*> → **STRING**
<*expression_boolenne*> → <*expression_boolenne*> **AND** <*expression_boolenne*>
<*expression_boolenne*> → <*expression_boolenne*> **OR** <*expression_boolenne*>
<*expression_boolenne*> → **!**<*expression_boolenne*>
<*expression_boolenne*> → <*expression_simple_booleenne*>
<*expression_boolenne*> → <*expression*> **>=** <*expression*>
<*expression_boolenne*> → <*expression*> **<=** <*expression*>
<*expression_boolenne*> → <*expression*> **>** <*expression*>
<*expression_boolenne*> → <*expression*> **<** <*expression*>
<*expression_boolenne*> → <*expression*> **==** <*expression*>
<*expression_simple_booleenne*> → **TRUE**
<*expression_simple_booleenne*> → **FALSE**
<*expression_simple_booleenne*> → **(**<*expression_booleenne*>**)**

4 Regex

Le fichier .flex contient des expressions régulières permettant de parser correctement le fichier d'entrée. Nous avons eu besoin de définir les regex suivantes :

— un nombre entier : ([1-9][0-9]*) | 0
— un nombre réel : (-|+) ? entier . ? [0-9]*([eE]- ?[0-9]+) ?
— des expressions booléennes : true|True|TRUE|#t et false|False|FALSE|#f
— un nom de variable en Java : [a-z][a-zA-Z0-9_]*
— une chaîne de caractères : " [^\n"]* "
— des commentaires : (("//"[^\n]*) | ("/*"[^"*/"]*"*/"))* | \n* | \r* | ' '* | \t*