Travaux dirigés

Introduction à la Compilation

2-Analyse syntaxique

Jean-Christophe Le Lann, Quentin Ducasse

Janvier 2023

Exercice 1 :

On donne ici la grammaire d'un mini-langage, que nous appellerons `Mini-C`.

program	:=	`int main ( ) {` *declaration** *statement** `}`
declaration	:=	*type identifier* (`[` *integer* `]`)? `;`
type	:=	`int` | `bool` | `float` | `char`
statement	:=	*assignment* | *if_statement* | *while_statement*
assignment	:=	*identifier* (`[` *expression* `]`)? `=` *expression* `;`
if_statement	:=	`if (` *expression* `) {` statement* `}` *else_statement?*
else_statement	:=	`else {` *statement** `}`
while_statement	:=	`while (` *expression* `) {` *statement** `}`
expression	:=	*conjunction* (`||` *conjunction*)*
conjunction	:=	*equality* (`&&` *equality*)*
equop	:=	`==` | `!=`
equality	:=	*relation* (*equop relation*)?
relation	:=	*addition* (*relop addition*)?
relop	:=	`<` | `<=` | `>` | `>=`
addition	:=	*term* (*addop term*)*
addop	:=	`+` | `-`
term	:=	*factor* (*mulop factor*)*
mulop	:=	`*` | `/` | `%`
factor	:=	*unaryop*? *primary*
unaryop	:=	`-` | `!`
primary	:=	*identifier* (`[` *expression* `]`) ? | *literal*|*parenth*
parenth	:=	`(` *expression* `)`

Notez que certains éléments lexicaux (lexèmes) n'ont pas été décrits ici :

`identifier`: qui peuvent contenir des minuscules, majuscules, chiffres et (*ex* : `nom_planete_1_EX`)

`literal : nombres entiers, nombres flottants, chaînes de caractères, true et false.`

Question 1 : Observer les règles de grammaire précédentes et déterminer les lexèmes du langage et leur type.

Question 2 : Ecrire quelques codes en `Mini-C`, qui soient corrects syntaxiquement (on ne demande pas d'écrire des codes qui aient un véritable sens algorithmique. Ces codes serviront en question 4 pour tester votre parseur.

Question 3 : Ecrire le lexeur de `Mini-C`. A partir d'un texte fourni en entrée (ou du nom du fichier contenant ce texte), le lexer doit générer un tableaux de lexèmes.

Nous cherchons maintenant à réaliser un *analyseur syntaxique* ou *parseur*. Son but est de consommer tous les lexèmes fournis par le lexeur : s'il parvient à consommer tous les lexèmes, on pourra considérer que texte analysé respecte les règles de grammaire du `Mini-C`. L'écriture de parseurs récursifs descendants est possible [1]. Nous vous livrons les fonctions suivantes, essentielles à l'analyse LL(k) :

- $\texttt{show}_n ext(k)$: $retournelek-imefuturlexmeprsentdansletableaudelexmessansl'enleverdelaliste.$`accept()` : $consommelelexmecourantenl'enlevantdelaliste.$
- `expect(token_kind)` : consomme le lexème courant s'il est du bon type (kind). Sinon retourne une erreur à l'utilisateur en indiquant la ligne et la colonne d'erreur.

Nou vous fournissons aussi une fonction `error` vous permettant de générer des erreurs de syntaxe. N'hésitez pas à vous en servir !

Question 4 : Compléter le code précédent de manière à parser les codes exemples écrits à la première question. Nous vous conseillons de rajouter des tests unitaires dans les fichiers de tests correspondants, à la fois pour le lexer et pour le parser.

N'essayez pas de parser un exemple qui contient l'intégralité de la grammaire, fonctionnez petit-à-petit avec des exemples de difficulté croissante et leurs tests associés

1. Elle est d'ailleurs assez fréquente, malgré la disponibilité d'outils de génération automatique.