CANAUD Frédéric - CAMPREDON Thomas

Projet Compilation

Construction d'un Analyseur Syntaxique pour un mini Langage C

Module S5 - Compilation | Langage : Python 3.9

Licence Informatique

4 janvier 2021

Professeurs : Karim Tamine

Sommaire

Liste des tâches réalisées 3

Structure de données 3

La grammaire est-elle LL(1) ? 5

Table d’analyse 7

Exécution du programme tableAnalyse.py 8

Exemples de chaînes 10

Liste des tâches réalisées

Durant ce projet réalisé en Python nous avons pu faire en sorte de :

- Calculer l'ensemble premier de tous les éléments non terminaux

- Calculer l'ensemble suivant de tous les éléments non terminaux

- Calculer la table d'analyse de la grammaire

- Déterminer si une chaîne est acceptée par la grammaire ou non

Structure de données

Grammaire

La grammaire proposée est décrite par la structure suivante :

- On utilise comme structure principal une **table de hachage** (HashMap), dont les **clés** sont tous les **éléments non terminaux** définissant une règle de production

- Chaque **clé** possède **une liste de règles de production**. Effectivement, si l'on prend par exemple l'élément non terminal "liste_declarations", deux règles de production lui sont associées. Or, on ne peut pas avoir deux clés dupliquées dans une table de hachage !

- Chaque **liste** contient une **liste** de symboles, composant la règle de production

Ainsi, la structure proposée est la suivante

```
regles = {
        "Programme": [["main(){", "liste_declarations", "liste_instructions", "}"]],
        "liste_declarations": [["une_declaration", "liste_declarations"], ["vide"]],
        "une_declaration": [["type", "id"]],
        "liste_instructions": [["une_instruction", "liste_instructions"], ["vide"]],
        "une_instruction": [["affectation"], ["test"]],
        "type": [["int"], ["float"]],
        "affectation": [["id", "=", "nombre", ";"]],
        "test": [["if", "condition", "une_instruction", "else", "une_instruction", ";"]],
        "condition": [["id", "operateur", "nombre"]],
        "operateur": [["<"], [">"], ["="]]
}
```

Ensemble premier et suivant

Un ensemble premier, ainsi qu'un ensemble suivant est décrit par la structure suivante :

- On utilise comme structure principal une **table de hachage** (HashMap), dont les **clés** sont tous les **éléments non terminaux** définissant une règle de production

- Chaque **clé** contient une **liste des éléments terminaux**, contenus dans les ensembles premiers ou suivants

Table d'analyse

La table d'analyse est définie par la structure suivante :

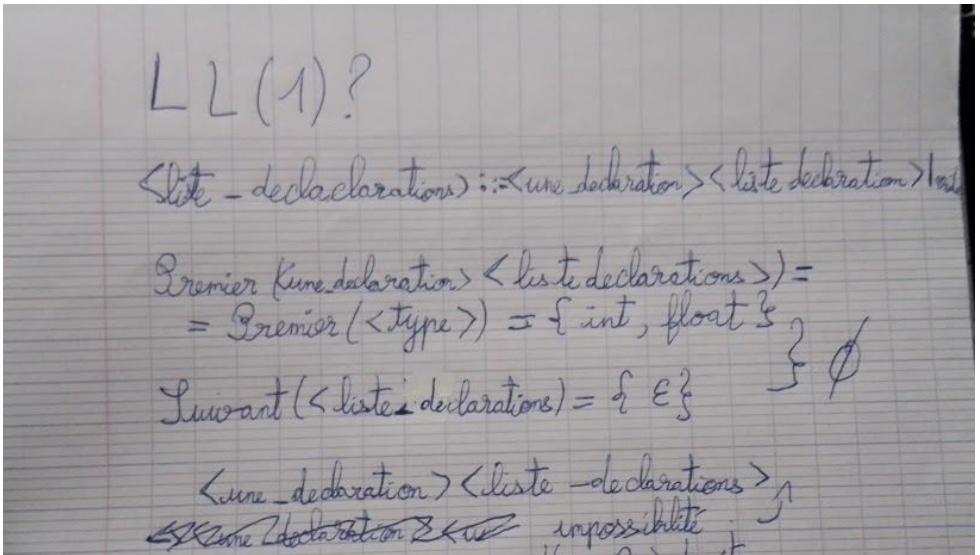
- On utilise comme structure principal une **table de hachage** (HashMap), dont les **clés** sont un couple composé :
    1. D'un **élément non terminal** définissant une règle de production
    2. D'un **élément terminal** contenu dans les ensembles premiers et suivants de la clé (différent du vide)
- Chaque **clé** est associée à une **règle de production**, c'est à dire par un **élément non terminal** associé à **une liste de symboles**

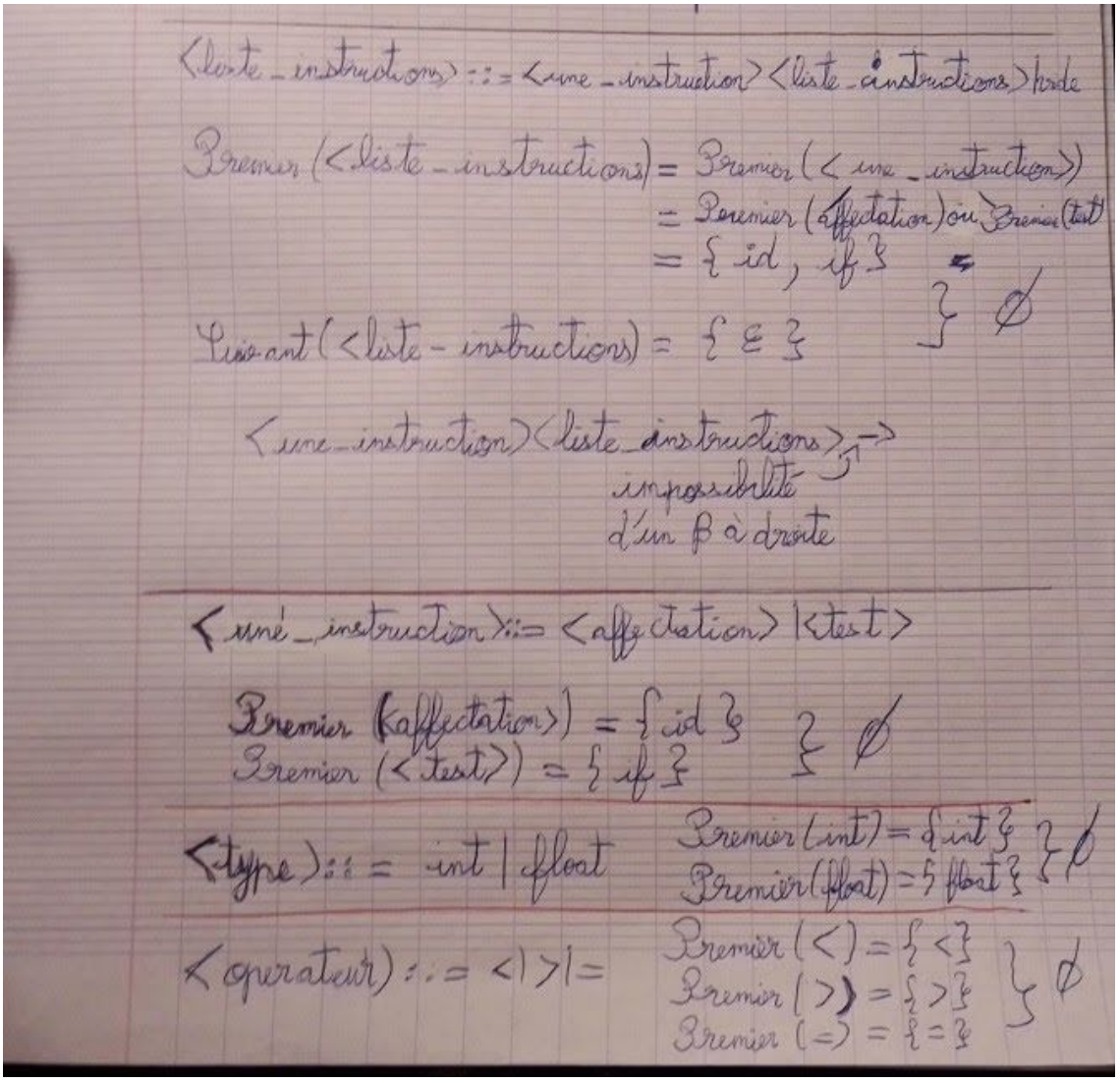
Usage : tableAnalyse[nonTerminal, ter] = {nonTerminal: production}

Exemple d'affichage de règles de la table d'analyse :

('une_declaration', 'int') : une_declaration -> type id

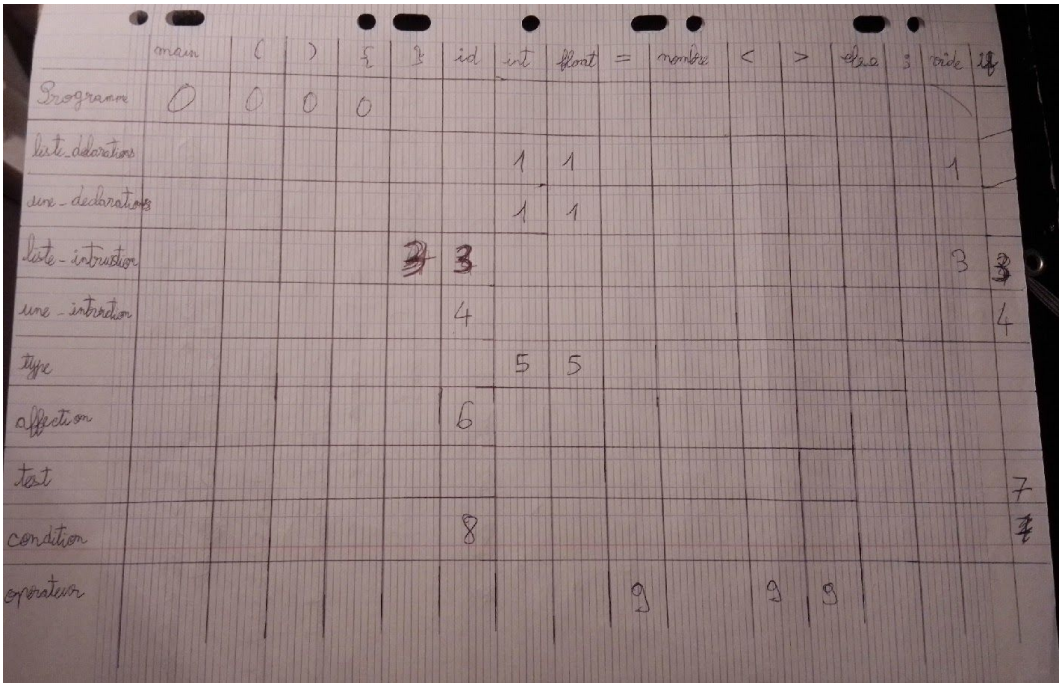
La grammaire est-elle LL(1) ?





Donc la grammaire est LL(1)

Table d'analyse



('Programme', 'main(){') : Programme-> main(){ liste_declarations liste_instructions }
('liste_declarations', 'int') : liste_declarations-> une_declaration liste_declarations
('liste_declarations', 'float') : liste_declarations-> une_declaration liste_declarations
('liste_declarations', '$') : liste_declarations-> vide
('liste_declarations', 'if') : liste_declarations-> vide
('liste_declarations', 'id') : liste_declarations-> vide
('une_declaration', 'int') : une_declaration-> type id
('une_declaration', 'float') : une_declaration-> type id
('liste_instructions', 'if') : liste_instructions-> une_instruction liste_instructions
('liste_instructions', 'id') : liste_instructions-> une_instruction liste_instructions
('liste_instructions', '}') : liste_instructions-> vide

('une_instruction', 'id') : une_instruction-> affectation
('une_instruction', 'if') : une_instruction-> test
('type', 'int') : type-> int
('type', 'float') : type-> float
('affectation', 'id') : affectation-> id = nombre ;
('test', 'if') : test-> if condition une_instruction else une_instruction ;
('condition', 'id') : condition-> id operateur nombre
('operateur', '<') : operateur-> <
('operateur', '>') : operateur-> >
('operateur', '=') : operateur-> =

Exécution du programme TableAnalyse.py

```
Les règles de production sont :

Programme ::= main(){ liste_declarations liste_instructions }  |
liste_declarations ::= une_declaration liste_declarations  | vide  |
une_declaration ::= type id  |
liste_instructions ::= une_instruction liste_instructions  | vide  |
une_instruction ::= affectation  | test  |
type ::= int  | float  |
affectation ::= id = nombre ;  |
test ::= if condition une_instruction else une_instruction ;  |
condition ::= id operateur nombre  |
operateur ::= <  | >  | =  |

Les ensembles premiers sont :

Premier( Programme ) = {'main(){'}
Premier( liste_declarations ) = {'int', 'vide', 'float'}
Premier( une_declaration ) = {'int', 'float'}
Premier( type ) = {'int', 'float'}
Premier( liste_instructions ) = {'vide', 'if', 'id'}
Premier( une_instruction ) = {'if', 'id'}
Premier( affectation ) = {'id'}
Premier( test ) = {'if'}
Premier( condition ) = {'id'}
Premier( operateur ) = {'=', '<', '>'}
```

```
Les ensembles suivants sont :

Suivant( Programme ) = {'$'}
Suivant( liste_declarations ) = {'$', 'if', 'id'}
Suivant( une_declaration ) = {'int', 'float'}
Suivant( liste_instructions ) = {'}'}
Suivant( une_instruction ) = {'else', ';', 'if', 'id'}
Suivant( type ) = {'id'}
Suivant( affectation ) = {'else', ';', 'if', 'id'}
Suivant( test ) = {'else', ';', 'if', 'id'}
Suivant( condition ) = {'if', 'id'}
Suivant( operateur ) = {'nombre'}

Éléments non terminaux :
 ['Programme', 'liste_declarations', 'une_declaration', 'liste_instructions', 'une_instruction', 'type', 'affectation', 'test', 'condition', 'operateur']

Éléments terminaux :
 ['main(){', '<', '$', ';', 'if', 'float', 'else', '=', '>', 'nombre', 'int', '}', 'id']
```

```
La table d'analyse est :

('Programme', 'main(){') : Programme-> main(){ liste_declarations liste_instructions }
('liste_declarations', 'int') : liste_declarations-> une_declaration liste_declarations
('liste_declarations', 'float') : liste_declarations-> une_declaration liste_declarations
('liste_declarations', '$') : liste_declarations-> vide
('liste_declarations', 'if') : liste_declarations-> vide
('liste_declarations', 'id') : liste_declarations-> vide
('une_declaration', 'int') : une_declaration-> type id
('une_declaration', 'float') : une_declaration-> type id
('liste_instructions', 'if') : liste_instructions-> une_instruction liste_instructions
('liste_instructions', 'id') : liste_instructions-> une_instruction liste_instructions
('liste_instructions', '}') : liste_instructions-> vide
('une_instruction', 'id') : une_instruction-> affectation
('une_instruction', 'if') : une_instruction-> test
('type', 'int') : type-> int
('type', 'float') : type-> float
('affectation', 'id') : affectation-> id = nombre ;
('test', 'if') : test-> if condition une_instruction else une_instruction ;
('condition', 'id') : condition-> id operateur nombre
('operateur', '<') : operateur-> <
('operateur', '>') : operateur-> >
('operateur', '=') : operateur-> =

Entrez l'expression à analyser (Séparez chaque symbole par un espace et finissez par $)
```

Exemples de chaînes

main(){ int id id = nombre; if id < nombre id = nombre; else id = nombre ; ; } ---> Générée par la grammaire

```
L'expression saisie est :  ['main(){', 'if', 'id', '<', 'nombre', 'id', '=', 'nombre', ';', 'else', 'id', '=', 'nombre', ';', ';', '}', '$']
['$', 'Programme']
['$', '}', 'liste_instructions', 'liste_declarations', 'main(){']
['$', '}', 'liste_instructions', 'liste_declarations']
['$', '}', 'liste_instructions', 'vide']
['$', '}', 'liste_instructions', 'une_instruction']
['$', '}', 'liste_instructions', 'test']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'condition', 'if']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'condition']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'nombre', 'operateur', 'id']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'nombre', 'operateur']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'nombre', '<']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'nombre']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'affectation']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', ';', 'nombre', '=', 'id']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', ';', 'nombre', '=']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', ';', 'nombre']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', ';']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else']
['$', '}', 'liste_instructions', ';', 'une_instruction']
['$', '}', 'liste_instructions', ';', 'affectation']
['$', '}', 'liste_instructions', ';', ';', 'nombre', '=', 'id']
['$', '}', 'liste_instructions', ';', ';', 'nombre', '=']
['$', '}', 'liste_instructions', ';', ';', 'nombre']
['$', '}', 'liste_instructions', ';', ';']
['$', '}', 'liste_instructions', ';']
['$', '}', 'liste_instructions']
['$', '}', 'vide']

Expression acceptée, générable par la grammaire
```

main(){ if id < nombre id = double; } $

```
L'expression saisie est :  ['main(){', 'if', 'id', '<', 'nombre', 'id', '=', 'double;', '}', '$']
['$', 'Programme']
['$', '}', 'liste_instructions', 'liste_declarations', 'main(){']
['$', '}', 'liste_instructions', 'liste_declarations']
['$', '}', 'liste_instructions', 'vide']
['$', '}', 'liste_instructions', 'une_instruction']
['$', '}', 'liste_instructions', 'test']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'condition', 'if']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'condition']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'nombre', 'operateur', 'id']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'nombre', 'operateur']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'nombre', '<']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction', 'nombre']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'une_instruction']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', 'affectation']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', ';', 'nombre', '=', 'id']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', ';', 'nombre', '=']
['$', '}', 'liste_instructions', ';', 'une_instruction', 'else', ';', 'nombre']

 Expression refusée, non générable par la grammaire
```