

Département des Sciences de  
 l’informatique

Rapport du projet de compilation

Objectif général :

L'objectif de ce projet est de Réaliser un compilateur en utilisant le langage JAVA en respectant les règles d’une grammaire choisie**.**

Elaboré par :

IDOUDI Med Aziz

Trabelsi Malek

Touj Seifeddine

- IGL3 –

Année universitaire : 2022-2023

# PRESENTATION DU LANGAGE

Le but est de réaliser un analyseur lexical, syntaxique et sémantique d’un sous ensemble du langage JAVA

## La structure du langage :

{

Déclaration

\*

\*

\*

FinDec

Corps

\*

\*

\*

FinCorps

}

## La Grammaire

S -> { Declaration D Corps I }

D -> id : T D | FinDec

I -> lire E I | ecrire E I | id ~~ E I | si E alors I | tantque E faire I | FinCorps

T-> char | entier | bolean | tableau [ nb ] T

E -> nb | id | ( E ) | id OPP id | nb OPP nb

OPP -> <= | >= | = | et | ou| +| -

## Analyse Lexicale

L'analyse lexicale est la première phase de la compilation. Dans le texte source, qui se présente comme un flot de caractères, l'analyse lexicale reconnaît des *unités lexicales*, qui sont les mots avec lesquels les phrases sont formées, et les présente à la phase suivante, l'analyse syntaxique.

Les principales sortes d'unités lexicales qu'on trouve dans les langages de programmation courants sont :

* Les identificateurs : id,a,etc ;
* Les caractères spéciaux simples : +, =, -, : ;
* Les caractères spéciaux doubles : <=, **><, >=** ,  ~~;
* Les mots-clés : si, alors, tantque,faire, les types;
* Les nombres : nb ,etc

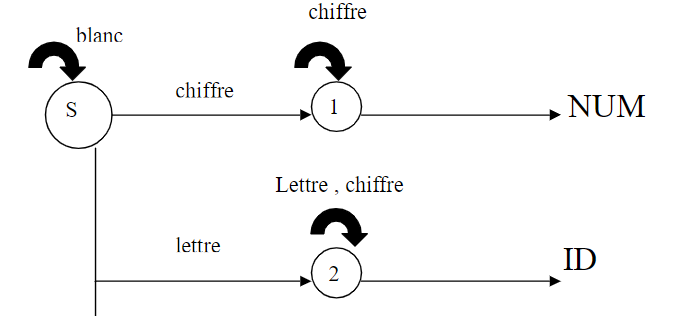
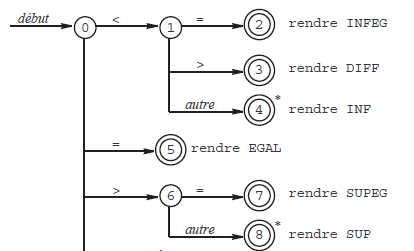
Les identificateurs et les nombres peuvent être reconnus grâce aux expressions régulières suivantes :

lettre → A | B | ... | Z | a | b | ... | z

chiffre → 0 | 1 | ... | 9

id → lettre ( lettre | chiffre )\*

chiffres → chiffre chiffre\*



## RECONNAISSANCE DES MOTS RÉSERVÉS

## . Les mots réservés appartiennent au langage défini par l'expression régulière lettre(lettre|chiffre)\*, tout comme les identificateurs. Leur reconnaissance peut donc se traiter de deux manières :

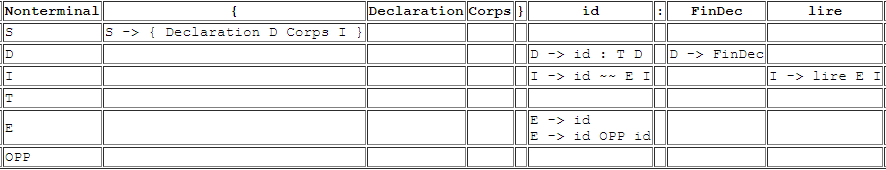
## on laisse l'analyseur traiter de la même manière les mots réservés et les identificateurs puis, quand la reconnaissance d'un « identificateur-ou-mot-réservé » est terminée , une fonction Verif vérifie s'il s'agit d'un identificateur ou d'un mot réservé.

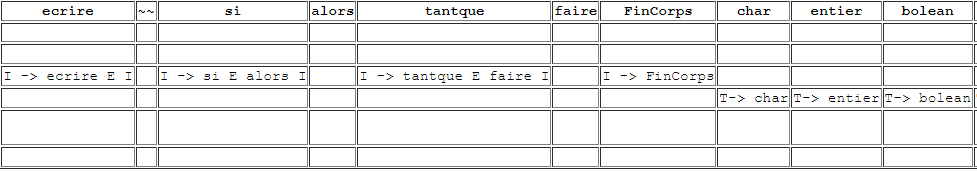
## ANALYSE SYNTAXIQUE

La deuxième phase (analyse syntaxique) consiste a expliciter la structure du programme sous forme d’un arbre de syntaxe.

**Type du grammaire**

Notre grammaire n’est pas de type LL(1) car la table contient des doublons comme montrez ci-dessous

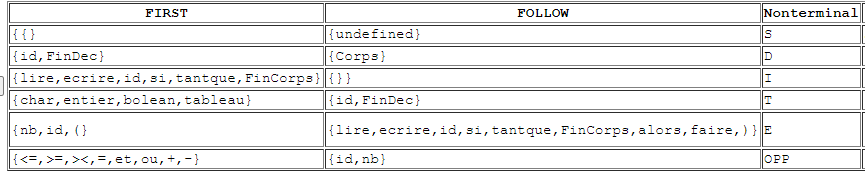




Table

Description automatically generated

Tel que les premiers et les suivants sont :



Donc, notre langage est de type SLR

1. **Augmentation de la grammaire**

S’-> S

S -> { Declaration D Corps I }

D -> id : T D | FinDec

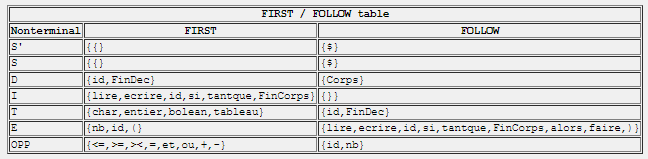
I -> lire E I | ecrire E I | id ~~ E I | si E alors I | tantque E faire I | FinCorps

T-> char | entier | bolean | tableau [ nb ] T

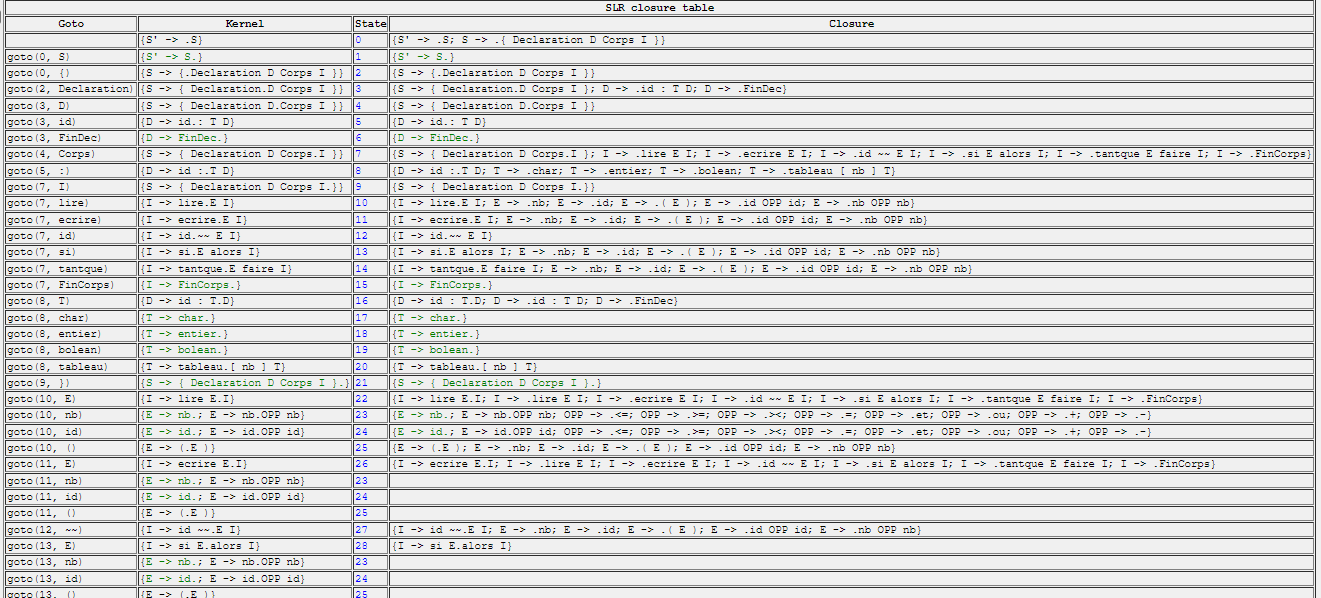
E -> nb | id | ( E ) | id OPP id | nb OPP nb

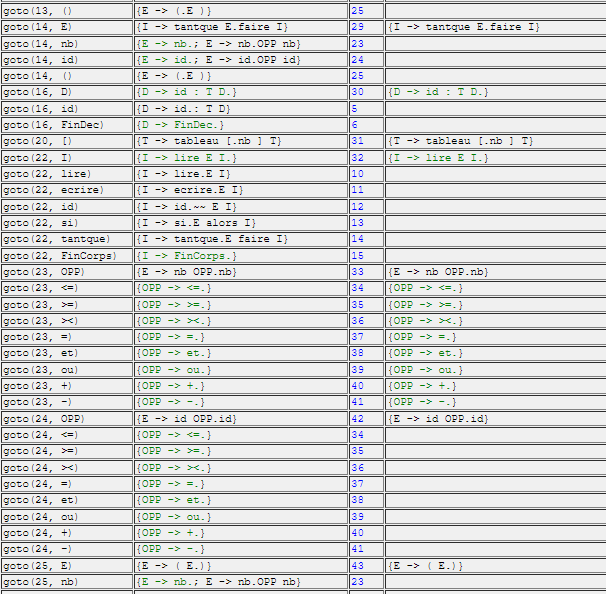
OPP -> <= | >= | = | et | ou| +| -

1. **Les premiers et les suivants :**

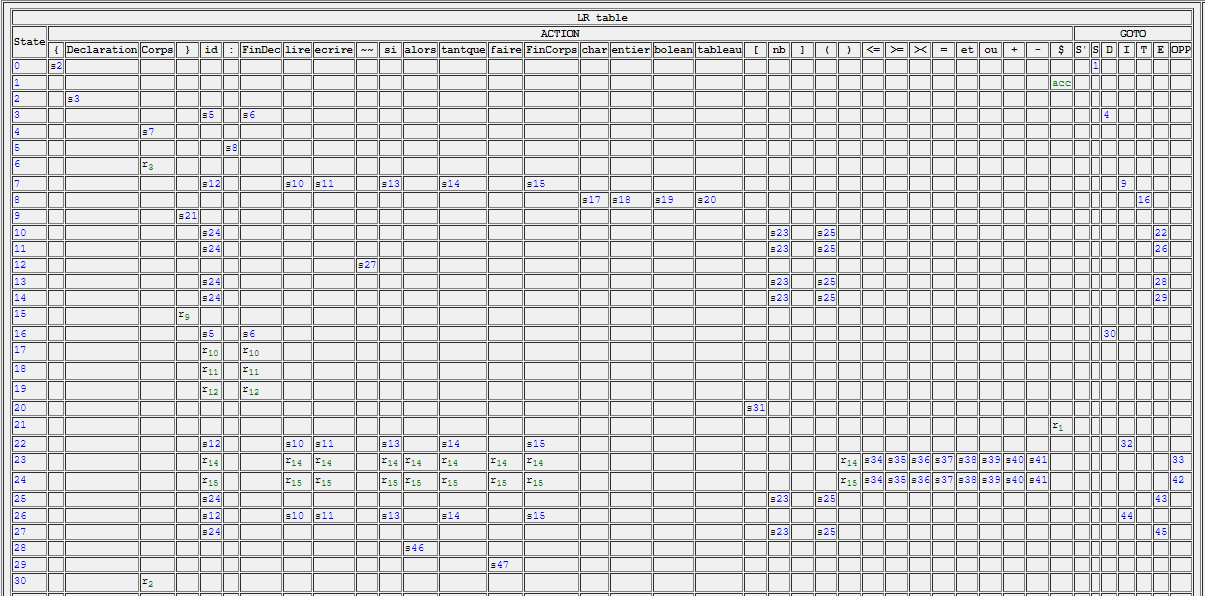
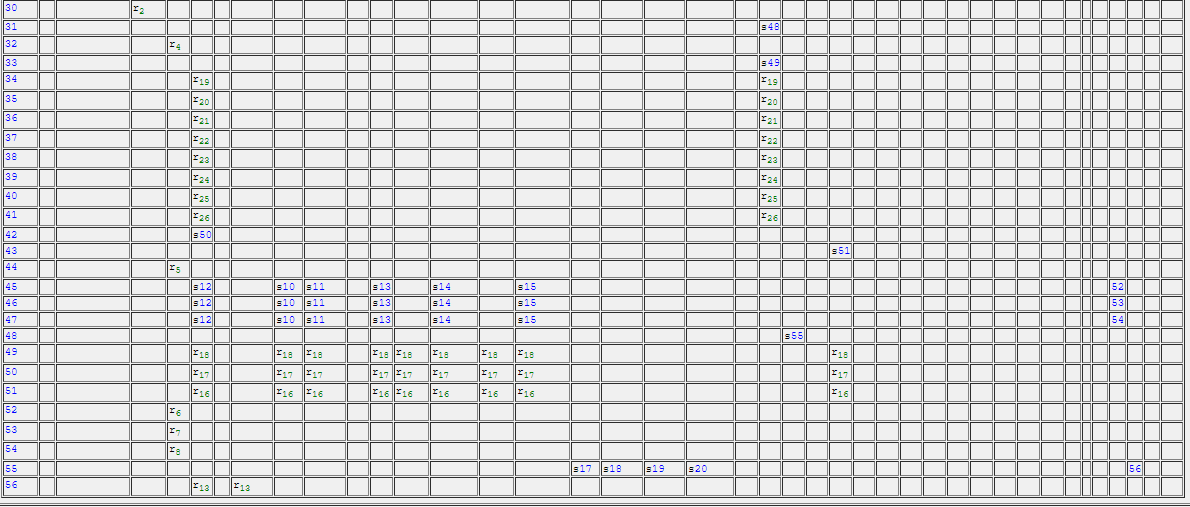
****

1. **La collection des items :**

****



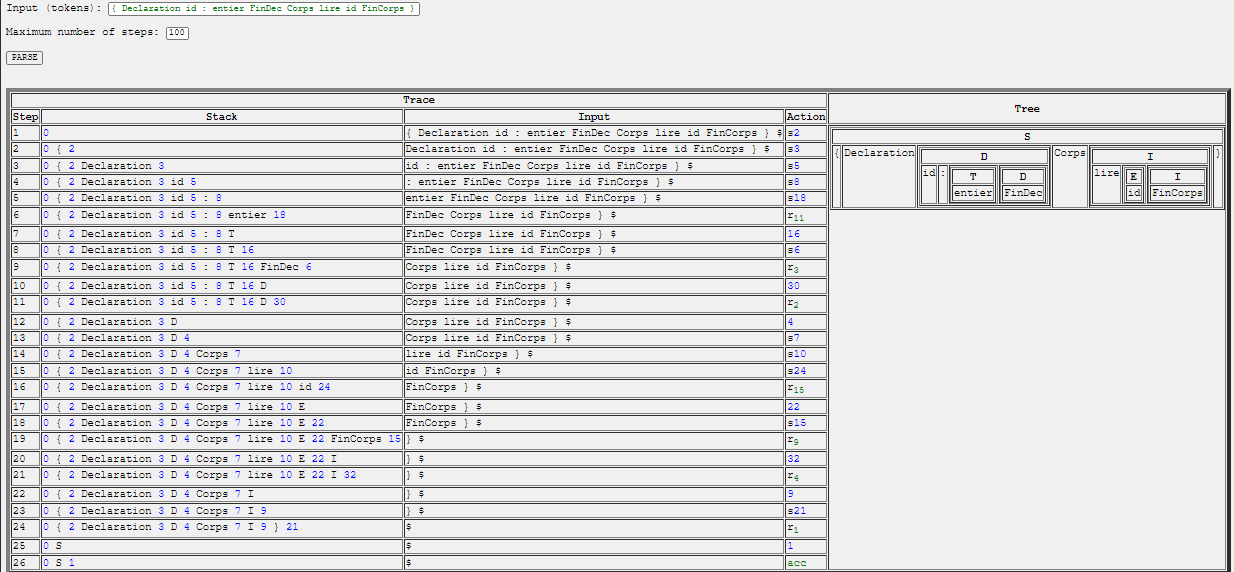
4-Table SLR



5-Exemple :

Le mot acceptée par notre langage

{ declaration a: entier FinDec Corps lire a ecrire a FinCorps }



**L’analyse sémantique**

1. **Générations de code**

|  |  |
| --- | --- |
| Règle de Production | Règle sémantique |
| S -> { déclaration D corps I } | s.code := D.code ||I.code || gen(S.place := ‘’{ Declaration ‘’, D.place ,’’corps’’, I.place ,’’}’’) |
| D-> FinDec | D.place := « FinDec » D.code := «  » |
| I-> lire E I | I.place :=nouvtemp() **I**.code :=E.code || I(1).code ||gen(‘lire’,E.place,I(1).place) |
| I-> ecrire E I | I.place :=nouvtemp()I.code :=E.code || I(1).code ||gen(‘ecrire’,E.place,I(1).place) |
| I->FinCorps | I.place := « FinCorps »D.code := «  » |
| E-> nb | I.place := nbD.code := «  » |
| E-> id | I.place := id.placeD.code := «  » |
| E->(E) | E.place :=nouvtemp() ; E.place :=E(1).place ;  E.code :=E(1).code ; |
| I-> id~~E I | I.code :=E.code||I.code||gen(id.place,’~~’,E.place,I.place |
| I-> si E alors I | I.place :=nouvtemp() I.ldebut :=nouvetiq()  I.lsuivant :=nouvetiq()  I.lvrai :=nouvetiq () ;  I.code :=gen(« I.debut »)||E.code||gen(‘si’,E.place ‘#’ 0 ‘goto ‘ I.lvrai’ )||gen(‘goto’,I.lsuivant) ; ||gen(I.lvrai’ :’)|| I(1).code ||gen(‘goto’,I.ldebut)|| gen(I.lsortie’ :’) |
| I-> tantque E faire I | I.place :=nouvtemp() I.ldebut :=nouvetiq()  I.lsuivant :=nouvetiq()  I.lvrai :=nouvetiq () ;  I.code :=gen(« I.debut »)||E.code||gen(‘si’,E.place ‘#’ 0 ‘goto’ ‘ I.lvrai’ )||gen(‘goto’,I.lsuivant) ; ||gen(I.lvrai’ :’)|| I(1).code ||gen(‘goto’,I.ldebut)|| gen(I.lsortie’ :’) |

## Le contrôle de type

|  |  |
| --- | --- |
| Production | Règle sémantique |
| S -> { Declaration D Corps I } | S.type := si I.type=vide et D.type=vide alors vide  Sinon erreur |
| I-> lire E I | I.type := si I(1).type= I.type alors vide |
| I-> ecrire E I |  |
| I-> nb OPP nb | si nb alors vrai sinon erreur\_de\_type |
| I-> id~~E I | si I(1).type=vide et E.type=id.type  alors vide sinon erreur\_de\_type |
| D-> id : T D | D.type := si D.type=vide et T.type=id.type alors vide sinon erreur |
| I-> tantque E faire I | I°.type :=si E°.type :=bolean alors I(1).type  Sinon erreur |
| I-> si E alors I | I°.type :=si E°.type :=bolean alors I(1).type  Sinon erreur |
| T-> char | T°.type := char |
| T->entier | T°.type :=entier |
| T-> bolean | T.type :=bolean |
| T->tab [nb] de T | T.type :=Tableau(1..nb.val , T(1).type) |
| E-> nb | E.type :=entier |
| E-> id | E.type :=Rechercher(id.entrée) |
| E-> id OPP id | si id(1).type=id(2).type alors vrai sinon  erreur\_de\_type |

## PROGRAMMES

Ce projet réalisé en java, est divisé en 5 classes :

UniteLexicale : classe où sont définis les Unité lexicale du langage.

Categorie : classe où sont définis les catégories du lexème

Scanner : classes où définit le traitement du flux du fichier de lecture.

ParserNew : classes où l’analyse syntaxique et sémantique sont réalisées

Main : classe permettant de boucler sur le traitement.

**Les captures d’exécution :**

## 

## 

## 