

**REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE**

**MINISTERE DE L’ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE**

**ECOLE NATIONALE SUPÉRIEURE D’INFORMATIQUE**

**Groupe : 2CS - SIQ1 Année : 2024/2025**

| **Travaux Pratiques du Module Compilation** |
| --- |

**Conception et Implémentation d’un Analyseur Syntaxique pour le Langage EXP avec JAVACC**

**Réalisé par :**

* FLISSI Loubna.
* LARBAOUI Yasmine Badr El Houda.

**Encadré par :**

Monsieur CHEBIEB AbdelKrim.

Table des matières

[**1. Spécifications du langage EXP : 3**](#_heading=h.e5tb8jmv3xtb)

[a. Alphabet et unités lexicales : 3](#_heading=h.6ssg8uml3png)

[b. Grammaire et règles de production : 3](#_heading=h.jycm02yieyj1)

[**2. Analyse syntaxique : 3**](#_heading=h.1blbe8f2rjpk)

[a. La grammaire est-elle LL(1) ? 3](#_heading=h.a671sm5ik72n)

[b. La table des Débuts et des Suivants : 5](#_heading=h.4nt3do5oct34)

[c. La table d’analyse : 6](#_heading=h.h8v14zcr6z3c)

[d. Résolution de la multi-définition : 7](#_heading=h.5biordf4kfoe)

[**3. L’arbre syntaxique du mot de EXP : =@id : 10**](#_heading=h.d6wj21xsevso)

# Spécifications du langage EXP :

## Alphabet et unités lexicales :

L’alphabet ∑ du langage EXP est composé de :

* Lettres et chiffres : A-Z, a-z, 0-9
* Symboles spéciaux : { } \_ > < ( ) + \* /

| **Le code de l’unité** | **Désignation de l’unité** | **Expression Régulière** |
| --- | --- | --- |
| **id** | Identifiants : suite de lettre et de chiffres commençant par une majuscule et terminant par \_ | [A-Z]([A-Za-z0-9])\*\_ |
| **$** | Les opérateurs : + \* / | [+ \* /] |
| **=** | Les opérateurs : > < | [> <] |
| **@** | Les délimiteurs : ( ) | [( )] |
| **&** | Les délimiteurs : { } | [{ }] |

## Grammaire et règles de production :

La grammaire G est définie comme suit : **G : < N, T, S, P >** avec :

P = S → A @ B | B & A | ε

A → B $ | S = S

B → A id | id S

# Analyse syntaxique :

## La grammaire est-elle LL(1) ?

Cette grammaire n'est pas adaptée à une analyse LL(1) car :

* Elle n’est pas propre car elle contient une epsilon production ( S → ε ), ce qui peut créer des ambiguïtés.
* Elle est récursive à gauche ( S → A → S ) , ( A → B → A ) , ( S → B → A → S ) ce qui empêche une analyse de gauche à droite sans transformation.
* Certaines productions ont des débuts qui se chevauchent, ce qui empêche de déterminer une seule règle à appliquer à partir du premier symbole lu.
* **Transformation de la grammaire :**

1. **Grammaire initiale :**

S → A @ B | B & A | ε ( epsilon production )

A → B $ | S = S (récursivité indirecte via S)

B → A id | id S (récursivité indirecte via A et S)

1. **Grammaire propre :**

Z → S | ε

S → A @ B | B & A

A → B $ | S = S | S = | = S | =

B → A id | id S | id

1. **Grammaire après l’élimination de la récursivité à gauche :**

| **Rappel :** L'élimination de la récursivité gauche.  Avant élimination : A → Aα | β  Après élimination : A → βA'  A' → αA' | ε |
| --- |

Z → S | ε

S → A @ B | B & A

A → B $ A' | B & A = S A' |B & A = A' | = S A' | = A'

A' → @ B = SA' | @ B = A' | ε

B → = S A' id B' | = A' id B' | id S B' | id B'

B' → $ A' id B' | & A = S A' id B' | & A = A' id B' | ε

1. **Grammaire finale après l’élimination de la récursivité à gauche et factorisation :**

| **Rappel :** La factorisation grammaticale.  Avant factorisation : A → αβ | αγ  Après factorisation : A → αA'  A' → β | γ |
| --- |

Z → S | ε

S → A @ B | B & A

A → B T | = X

A' → @ B = X | ε

T → $ A' | & A = X

X → S A' | A'

B → = Y | id F

B' → $ A' id B' | & A = Y | ε

F → S B' | B'

Y → S A' id B' | A' id B'

## La table des Débuts et des Suivants :

Avant de construire la table d'analyse de notre grammaire, il faut d'abord déterminer les ensembles Débuts et Suivants pour chaque non-terminal. Ces ensembles sont essentiels pour identifier les symboles pouvant apparaître en début ou en fin de dérivation.

Voici la table des ensembles Débuts et Suivants de la grammaire :

| **S0 → Z#**  Z → S | ε  S → A @ B | B & A  A → B T | = X  A' → @ B = X | ε  T → $ A' | & A = X  X → S A' | A'  B → = Y | id F  B' → $ A' id B' | & A = Y | ε  F → S B' | B'  Y → S A' id B' | A' id B' |
| --- |

| **Symbole** | **Débuts** | **Suivants** |
| --- | --- | --- |
| **Z** | id = | # |
| **S** | id = | @ id & = $ # |
| **A** | id = | @ id & = $ # |
| **A**' | @ ε | @ id & = $ # |
| **T** | $ & | @ id & = $ # |
| **X** | id = @ ε | @ id & = $ # |
| **B** | id = | @ id & = $ # |
| **B**' | $ & ε | @ id & = $ # |
| **F** | id = $ & ε | @ id & = $ # |
| **Y** | id = @ ε | @ id & = $ # |

## La table d’analyse :

Avant de construire la table d’analyse, nous remplaçons tous les MDP par un terminal dans toutes les productions, puis nous attribuons un numéro à chaque production de la grammaire:

| **N°P** | **La production** | **N°P** | **La production** |
| --- | --- | --- | --- |
| **1** | Z → S | **12** | X → A' |
| **2** | Z → ε | **13** | B → = Y |
| **3** | S → A @ B | **14** | B → id F |
| **4** | S → B & A | **15** | B' → $ A' id B' |
| **5** | A → B T | **16** | B' → & A = Y |
| **6** | A → = X | **17** | B' → ε |
| **7** | A' → @ B = X | **18** | F → S B' |
| **8** | A' → ε | **19** | F → B' |
| **9** | T → $ A' | **20** | Y → S A' id B' |
| **10** | T → & A = X | **21** | Y → A' id B' |
| **11** | X → S A' | **22** |  |

Voici la table d’analyse de la grammaire après transformation :

| **Symbole** | **id** | **=** | **@** | **$** | **&** | **#** |
| --- | --- | --- | --- | --- | --- | --- |
| **Z** | 1 | 1 |  |  |  | 2 |
| **S** | 3, 4 | 3, 4 |  |  |  |  |
| **A** | 5 | 5, 6 |  |  |  |  |
| **A**' | 8 | 8 | 7, 8 | 8 | 8 | 8 |
| **T** |  |  |  | 9 | 10 |  |
| **X** | 11, 12 | 11, 12 | 12 | 12 | 12 | 12 |
| **B** | 14 | 13 |  |  |  |  |
| **B**' | 17 | 17 | 17 | 15, 17 | 16, 17 | 17 |
| **F** | 18, 19 | 18, 19 | 19 | 19 | 19 | 19 |
| **Y** | 20, 21 | 20, 21 | 21 | 21 | 21 | 21 |

La table d’analyse est multi-définie, car plusieurs cases contiennent plus d’une production. Cela signifie que la grammaire n'est toujours pas LL(1) et qu’elle reste ambiguë, rendant l’analyse non déterministe. Il est donc nécessaire de la modifier pour qu’une seule production soit choisie à chaque étape.

## Résolution de la multi-définition :

Pour résoudre le problème de la multi-définition et rendre la construction de l'analyse syntaxique de notre grammaire possible, plusieurs méthodes peuvent être appliquées :

Tout d'abord, il faut vérifier si la grammaire peut encore être transformée pour devenir LL(1). Si elle ne l'est toujours pas, plusieurs solutions sont envisageables :

* **Vérifier si la grammaire peut être LL(k) pour k >1** en regardant plus de symboles en entrée. Cela complexifie l’analyse mais peut résoudre certaines ambiguïtés.
* **Modifier la grammaire pour rendre la table d’analyse mono-définie** en éliminant la récursivité gauche, en factorisant les règles et en imposant des conventions pour restreindre certaines productions.
* **Changer complètement la grammaire :** Si aucune des méthodes précédentes ne permet d’obtenir une grammaire compatible avec LL(1), il peut être nécessaire de la reformuler entièrement.

Voici notre grammaire G1 :  **< N, T, Z, P > :**

| Z → S | ε  S → A @ B | B & A  A → B T | = X  A' → @ B = X | ε  T → $ A' | & A = X  X → S A' | A'  B → = Y | id F  B' → $ A' id B' | & A = Y | ε  F → S B' | B'  Y → S A' id B' | A' id B' |
| --- |

Après avoir vérifié notre grammaire et incrémenté le paramètre k, nous avons utilisé un logiciel pour tester si la grammaire est LL(k). Malgré les transformations appliquées, telles que l’élimination de la récursion gauche et la factorisation, la grammaire n’a pas satisfait les critères LL(k) pour k>1.

Cependant, nous avons décidé d’appliquer certaines conventions, telles que la priorité des productions précises et courtes, ainsi que la consommation des terminaux avant les non-terminaux, tout en conservant la structure fondamentale de la grammaire intacte. Ces conventions ont permis de simplifier certaines règles et de mieux structurer la grammaire, facilitant ainsi l’analyse syntaxique.

**Conventions appliquées :**

Les conventions suivantes sont utilisées pour rendre la grammaire LL(1) claire et permettre au parseur de faire des choix précis avec un seul symbole à chaque étape :

| **Convention** | | **Application** |
| --- | --- | --- |
| **1** | Priorité des opérateurs | @ : priorité la plus élevée  & : priorité moyenne  = : priorité faible |
| **2** | L'associativité des opérateurs | & : associatif à gauche  = : associatif à droite |
| **3** | L'ajout de ε-productions  22 : T → ε  23 : Y → ε | Pour résoudre les conflits des débuts/ suivants ( 22, 23 ) |

**La table d’analyse après application des conventions :**

| **Symbole** | **id** | **=** | **@** | **$** | **&** | **#** |
| --- | --- | --- | --- | --- | --- | --- |
| **Z** | 1 | 1 |  |  |  | 2 |
| **S** | 3 | 3 |  |  | 4 |  |
| **A** | 5 | 5 |  |  |  |  |
| **A**' | 8 | 8 | 7 | 8 | 8 | 8 |
| **T** |  |  |  | 9 | 10 | 22 |
| **X** | 11 | 11 | 12 | 12 | 12 | 12 |
| **B** | 14 | 13 |  |  |  |  |
| **B**' | 17 | 17 | 17 | 15 | 16 | 17 |
| **F** | 18 | 18 | 19 | 19 | 19 | 19 |
| **Y** | 20 | 20 | 21 | 21 | 21 | 23 |

Après application des conventions, la table est mono-définie donc la grammaire est LL(1), et maintenant on peut construire l’analyseur syntaxique correctement.

# L’arbre syntaxique du mot de EXP : =@id :

