

Projet Théorie de compilation

2GI

Projet C-Pascal

*Réalisé par :* *Encadrant :*

- OUTIDRARINE Mohamed - Pr. Souhail GHAZI

- ECH-CHEBLAOUI Yassine

- ZAHAD Zakaria

-AMGHAR Souhail

Sommaire:

[***1- Alphabet du langage : 3***](#_Toc25353078)

[***2- Mots clés : 3***](#_Toc25353079)

[***3- Grammaire et Intégration de l’option 1 : 3***](#_Toc25353080)

[***4- Grammaire LL(1) 5***](#_Toc25353081)

[*** Eliminer la récursivité à gauche de la grammaire : 5***](#_Toc25353082)

[*** Factorisation de la grammaire : 6***](#_Toc25353083)

[*** Grammaire Après Changement : 8***](#_Toc25353084)

[*** Vérification de la grammaire: 10***](#_Toc25353085)

[***5- Version finale de grammaire LL(1) 17***](#_Toc25353086)

# 1- Alphabet du langage :

Alphabet= {a, …, z, A, …, Z, 0, …, 9, , , ; , = ,< , >, !, +, -, \* , /, |, &, {, }, [, ], (, ), #, ’ }

# 2- Mots clés :

Mot clé= {main ,entier, car, si, alors, sinon, tantque, faire, ecrire, lire, retour}

# 3- Grammaire et Intégration de l’option 1 :

L’option 1 consiste à intégrer dans la grammaire le type caractère.

Après l’intégration de cette structure, nous avons obtenu la grammaire

suivante :

1 ) <programme> =:: <liste de déclarations> <liste de fonctions>

2 ) <liste de fonctions> =:: <fonction> <liste de fonctions> | main() { <liste d'instructions> }

3 ) <fonction>= ::< identificateur>(<liste de paramètres>)<liste de déclarations>{ <liste d'instructions> }

4 ) <liste de déclarations>= ::< déclarations> ; | 𝜀

5 ) < déclarations> =:: <déclaration> | <déclarations> , <déclaration>

6 ) <déclaration> =:: entier <identificateur>

| Car <identificateur>

| entier <identificateur> [<expression>]

| Car <identificateur> [<expression>]

7 ) <liste de paramètres>= :: 𝜀 | <paramètres>

8 ) < paramètres> =:: <paramètre> | < paramètres>,<paramètre>

9 ) <paramètre> =:: entier <identificateur> | Car <identificateur>

10 ) <liste d'instructions> =:: 𝜀 | <instruction> ; <liste d'instructions>

11 ) <instruction> =:: <expression>

|<identificateur>=<expression>

| <identificateur>[<expression>] =<expression>

| retour <expression>

| si <expression> alors { <liste d’instructions> } sinon { < liste d’instructions > }

| si <expression> alors { < liste d’instructions > }

| tantque (<expression>) faire { < liste d’instructions > }

| ecrire( <expression> )

| <identificateur> = lire()

| <identificateur>[<expression>] = lire()

12 ) <expression> =:: <expression> <operateur logique> <expression logique>

| <expression logique>

13 ) <expression logique> =:: <expression logique > < comparaison> <expression simple>

| <expression simple>

14 ) <expression simple> =:: <expression simple>+<terme>

| <expression simple>-<terme>

| <terme>

| -<terme>

15 ) <terme> =:: <terme>\* <facteur>

| <terme>/ <facteur>

|<terme prioritaire>

16 ) <terme prioritaire>=::!<facteur>|<facteur>

17 ) <facteur> =:: <identificateur>

| <identificateur>(< paramètres effectifs >)

| <cste>

| (<expression>)

| <identificateur>[<expression>]

|’<lettre>’

18 ) < paramètres effectifs > =:: 𝜀 | <expressions>

19 ) <expressions> =:: <expression>| <expressions> , <expression>

20 ) <operateur logique> =:: | | &

21 ) <comparaison> =:: <|>| == | <= | >= | !=

22 ) <identificateur> =:: <lettre> <mot>

23 ) <mot> = :: 𝜀 | <lettre><mot> | <chiffre><mot>

24 ) <cste> =:: <chiffre> | <chiffre><cste>

25 ) <chiffre> =:: 0|1|…|8|9

26 ) <lettre> =:: A|B|…|Z|a|b|…|z

# 4- Grammaire LL(1)

Dans cette partie, on va d’abord éliminer la récursivité { gauche de la grammaire, puis la factoriser et après vérifier qu’elle est une grammaire LL(1).

## Eliminer la récursivité à gauche de la grammaire :

Dans notre grammaire, la récursivité est éliminée sur la majorité des règles sauf :

5 ) < déclarations> =:: <déclaration> | <déclarations> , <déclaration>

8 ) < paramètres> =:: <paramètre> | < paramètres>,<paramètre>

12 ) <expression> =:: <expression> <operateur logique> <expression logique>

| <expression logique>

13 ) <expression logique> =:: <expression logique > < comparaison> <expression simple>

| <expression simple>

14 ) <expression simple> =:: <expression simple>+<terme>

| <expression simple>-<terme>

| <terme>

| -<terme>

15 ) <terme> =:: <terme>\* <facteur>

| <terme>/ <facteur>

|<terme prioritaire>

19 ) <expressions> =:: <expression>| <expressions> , <expression>

Alors en appliquant les règles d’élimination de la récursivité à gauche on trouve :

5 ) < déclarations> =:: <déclaration> <déclarations ‘>

5 ) <déclarations ‘> = ::  , <déclaration> <déclarations ‘> | 𝜀

8 ) < paramètres> =:: <paramètre> < paramètres ‘>

8 ) < paramètres ’> =:: , <paramètre> <paramètres ‘> | 𝜀

12 ) <expression> =:: <expression logique> <expression ‘>

12 ) <expression ‘> =:: <operateur logique> <expression logique> <expression ‘> | 𝜀

13 ) <expression logique> =:: <expression simple><expression logique ‘>

13 ) <expression logique ‘> =:: < comparaison> <expression simple> <expression logique ‘>| 𝜀

14 ) <expression simple> =:: <terme><expression simple ‘> | -<terme><expression simple ‘>

14 ) <expression simple ‘> =:: +<terme><expression simple ‘>

| -<terme><expression simple ‘>

| 𝜀

15 ) <terme> =:: <terme prioritaire> <terme ‘>

15 ) <terme ‘> =:: \*<facteur><terme ‘>

| /<facteur><terme ‘>

| 𝜀

19 ) <expressions> =:: <expression> <expressions ‘>

19 ) <expressions ‘> =:: , <expression> <expressions ‘>| 𝜀

## Factorisation de la grammaire :

Dans notre grammaire, on a des problèmes de factorisation sur les règles suivantes :

6 ) <déclaration> =:: entier <identificateur>

| Car <identificateur>

| entier <identificateur> [<expression>]

| Car <identificateur> [<expression>]

11 ) <instruction> =:: <expression>

|<identificateur>=<expression>

| <identificateur>[<expression>] =<expression>

| retour <expression>

| si <expression> alors { <liste d’instructions> } sinon { < liste d’instructions > }

| si <expression> alors { < liste d’instructions > }

| tantque (<expression>) faire { < liste d’instructions > }

| ecrire( <expression> )

| <identificateur> = lire()

| <identificateur>[<expression>] = lire()

17 ) <facteur> =:: <identificateur>

| <identificateur>(< paramètres effectifs >)

| <cste>

| (<expression>)

| <identificateur>[<expression>]

|’<lettre>’

24 ) <cste> =:: <chiffre> | <chiffre><cste>

Alors en appliquant les règles de factorisation à gauche on trouve :

6 ) <déclaration> =:: entier <déclaration ‘> | Car <déclaration ‘>

6 ) <déclaration ‘> =:: <identificateur><déclaration ‘’>

6 ) <déclaration ‘’> =:: 𝜀 | [<expression>]

11 ) <instruction> =:: <expression>

|<identificateur> <instruction ‘>

| retour <expression>

| si <expression> alors { <liste d’instructions> } <instruction ‘’>

| tantque (<expression>) faire { < liste d’instructions > }

| ecrire( <expression> )

11 ) <instruction ‘> =:: =<instruction ‘’’> | [<expression>]= <instruction ‘’’>

11 ) <instruction ‘’’> = :: lire() | <expression>

11 ) <instruction ‘‘> =:: sinon { <liste d’instructions> } | 𝜀

17 ) <facteur> =:: <identificateur> <facteur ‘>

| <cste>

| (<expression>)

| ’<lettre>’

17 ) <facteur ‘> =:: 𝜀 | [ <expression> ] | (< paramètres effectifs >)

24 ) <cste> =:: <chiffre><cste ‘>

24 ) <cste ‘> =:: 𝜀 | <cste>

## Grammaire Après Changement :

1 ) <programme> =:: <liste de déclarations> <liste de fonctions>

2 ) <liste de fonctions> =:: <fonction> <liste de fonctions> | main() { <liste d'instructions> }

3 ) <fonction>= ::< identificateur>(<liste de paramètres>)<liste de déclarations>{ <liste d'instructions> }

4 ) <liste de déclarations>= ::< déclarations> ; | 𝜀

5 ) < déclarations> =:: <déclaration> <déclarations ‘>

5 ) <déclarations ‘> = :: , <déclaration> <déclarations ‘> | 𝜀

6 ) <déclaration> =:: entier <déclaration ‘> | Car <déclaration ‘>

6 ) <déclaration ‘> =:: <identificateur><déclaration ‘’>

6 ) <déclaration ‘’> =:: 𝜀 | [<expression>]

7 ) <liste de paramètres>= :: 𝜀 | <paramètres>

8 ) < paramètres> =:: <paramètre> < paramètres ‘>

8 ) < paramètres ’> =:: , <paramètre> <paramètres ‘> | 𝜀

9 ) <paramètre> =:: entier <identificateur> | Car <identificateur>

10 ) <liste d'instructions> =:: 𝜀 | <instruction> ; <liste d'instructions>

11 ) <instruction> =:: <expression>

|<identificateur> <instruction ‘>

| retour <expression>

| si <expression> alors { <liste d’instructions> } <instruction ‘’>

| tantque (<expression>) faire { < liste d’instructions > }

| ecrire( <expression> )

11 ) <instruction ‘> =:: =<instruction ‘’’> | [<expression>]= <instruction ’’’>

11 ) <instruction ‘’’> = :: lire() | <expression>

11 ) <instruction ‘‘> =:: sinon { <liste d’instructions> } | 𝜀

12 ) <expression> =:: <expression logique> <expression ‘>

12 ) <expression ‘> =:: <operateur logique> <expression logique> <expression ‘> | 𝜀

13 ) <expression logique> =:: <expression simple><expression logique ‘>

13 ) <expression logique ‘> =:: < comparaison> <expression simple> <expression logique ‘>| 𝜀

14 ) <expression simple> =:: <terme><expression simple ‘> | -<terme><expression simple ‘>

14 ) <expression simple ‘> =:: +<terme><expression simple ‘>

| -<terme><expression simple ‘>

| 𝜀

15 ) <terme> =:: <terme prioritaire> <terme ‘>

15 ) <terme ‘> =:: \*<facteur><terme ‘>

| /<facteur><terme ‘>

| 𝜀

16 ) <terme prioritaire>=::!<facteur>|<facteur>

17 ) <facteur> =:: <identificateur> <facteur ‘>

| <cste>

| (<expression>)

| ’<lettre>’

17 ) <facteur ‘> =:: 𝜀 | [ <expression> ] | (< paramètres effectifs >)

18 ) < paramètres effectifs > =:: 𝜀 | <expressions>

19 ) <expressions> =:: <expression> <expressions ‘>

19 ) <expressions ‘> =:: , <expression> <expressions ‘>| 𝜀

20 ) <operateur logique> =:: | | &

21 ) <comparaison> =:: <|>| == | <= | >= | !=

22 ) <identificateur> =:: <lettre> <mot>

23 ) <mot> = :: 𝜀 | <lettre><mot> | <chiffre><mot>

24 ) <cste> =:: <chiffre><cste ‘>

24 ) <cste ‘> =:: 𝜀 | <cste>

25 ) <chiffre> =:: 0|1|…|8|9

26 ) <lettre> =:: A|B|…|Z|a|b|…|z

## Vérification de la grammaire:

On teste si la grammaire est LL(1) ou non par la propriété suivante :

#### Propriété :



Une grammaire est LL(1) si pour tout non-terminal X apparaissant dans le membre

gauche de deux productions :

Xa

Alors :

,

Xb

1. Premier(a) INTERSECTION Premier(b) égale à l’ensemble vide.
2. Une des conditions suivante est vraie :
   * Ni a ni b n’est annulable et aucune ne se dérive en epsilon.
   * Uniquement, a ou bien b est annulable et Premier(X) INTERSECTION Suivant(X) égale à l’ensemble vide.

Premièrement, on calculera les Premiers et Les Suivants de tous les productions de la grammaire

#### Table d’analyse :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Non terminal** | | **Premier** | | **Suivant** | |
| **<programme>** | | Entier, car , 𝜀 | | $ | |
| **<liste de fonctions >** | | lettre, main , 𝜀 | | $ | |
| **<fonction >** | | Lettre | | -premier <liste de fonctions> | |
| **<liste de déclarations >** | | Entier , car , 𝜀 | | -premier <liste de fonctions>  { | |
| **<déclarations >** | | Entier , car | | ; | |
| **<déclarations ’>** | | , , 𝜀 | | ; | |
| **<déclaration >** | | Entier , car | | , , ; | |
| **<déclaration ‘>** | | Lettre | | , , ; | |
| **<déclaration ‘’>** | | [, 𝜀 | | , , ; | |
| **<liste de paramètres >** | | Entier , car , 𝜀 | | ) | |
| **<paramètres >** | | Entier , car | | ) | |
| **<paramètres ‘>** | | , , 𝜀 | | ) | |
| **<paramètre >** | | Entier , car | | , , ) | |
| **<liste d’instructions >** | | -premier <expression>  -Lettre , retour , si , tantque , ecrire , 𝜀 | | } | |
| **<instruction>** | | -premier <expression>  -Lettre , retour , si , tantque , ecrire | | ; | |
| **<instruction ‘>** | | =, [ | | ; | |
| **<instruction ‘’>** | | sinon, 𝜀 | | ; | |
| **<instruction ‘’’>** | | -premier <expression>  -lire | | ; | |
| **<expression>** | | -premier <expression simple> | | , , alors , faire , ] , } ,  ; , ) | |
| **<expression ‘>** | | &, | , 𝜀 | | , , alors , faire , ] , } ,  ; , ) | |
| **<expression logique>** | | -premier <expression simple> | | &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<expression logique ‘>** | | <, >, ==, <=, >=, !=, 𝜀 | | &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<expression simple>** | | Lettre, chiffre, ( , ‘ , ! , - | | <, >, ==, <=, >=, !=  &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<expression simple ‘>** | | + , - , 𝜀 | | <, >, ==, <=, >=, !=  &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<terme>** | | ! , Lettre, chiffre, ( , ‘ | | + , -  <, >, ==, <=, >=, !=  &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<terme ‘>** | | \* , / , 𝜀 | | + , -  <, >, ==, <=, >=, !=  &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<terme prioritaire>** | | ! , Lettre, chiffre, ( , ‘ | | \* , /  + , -  <, >, ==, <=, >=, !=  &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<facteur>** | | Lettre, Chiffre, ( , ‘ | | \* , /  + , -  <, >, ==, <=, >=, !=  &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<facteur ‘>** | | [ , 𝜀 , ( | | \* , /  + , -  <, >, ==, <=, >=, !=  &, | , , , alors , faire , ] , } ,  ; , ) | |
| **<paramètres effectifs>** | | \*,/ , 𝜀 | | ) | |
| **<expressions>** | | -premier <expression> | | ) | |
| **<expressions ‘>** | | , , 𝜀 | | ) | |
| **<operateur logique >** | | | , & | | -premier <expression logique> 🡺 Lettre, chiffre, ( , ‘ , ! , - | |
| **<comparaison>** | | <, >, ==, <=, >=, != | | Lettre, chiffre, ( , ‘ , ! , - | |
| **<identificateur>** | | Lettre | | * Premier et suivant <facteur ‘> * Premier <instruction ‘> * Suivant <paramètre> * Premier <déclaration ‘’>   ( | |
| **<mot>** | | Lettre, chiffre , 𝜀 | | Suivant <identificateur> | |
| **<cste>** | | Chiffre | | Suivant <facteur> | |
| **<cste ‘>** | | Chiffre , 𝜀 | | Suivant <facteur> | |

Vérification des productions :

1 ) <programme> =:: <liste de déclarations> <liste de fonctions>

**🡺 La production est LL(1)**

2 ) <liste de fonctions> =:: <fonction> <liste de fonctions> | main() { <liste d'instructions> }

**🡺 La production est LL(1) (Pr(**fonction**)ᴖPr**(main)**=Φ )**

3 ) <fonction>= ::< identificateur>(<liste de paramètres>)<liste de déclarations>{ <liste d'instructions> }

**🡺 La production est LL(1)**

4 ) <liste de déclarations>= ::< déclarations> ; | 𝜀

**🡺 La production est LL(1) (Pr(**déclarations**)ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

5 ) < déclarations> =:: <déclaration> <déclarations ‘>

**🡺 La production est LL(1)**

5 ) <déclarations ‘> = :: , <déclaration> <déclarations ‘> | 𝜀

**🡺 La production est LL(1) (Pr( , )ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

6 ) <déclaration> =:: entier <déclaration ‘> | Car <déclaration ‘>

**🡺 La production est LL(1)**

6 ) <déclaration ‘> =:: <identificateur><déclaration ‘’>

**🡺 La production est LL(1)**

6 ) <déclaration ‘’> =:: 𝜀 | [<expression>]

**🡺 La production est LL(1) (Pr(** [ **)ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

7 ) <liste de paramètres>= :: 𝜀 | <paramètres>

**🡺 La production est LL(1) (Pr(**paramètres**)ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

8 ) < paramètres> =:: <paramètre> < paramètres ‘>

**🡺 La production est LL(1)**

8 ) < paramètres ’> =:: , <paramètre> <paramètres ‘> | 𝜀

**🡺 La production est LL(1) (Pr(** , **)ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

9 ) <paramètre> =:: entier <identificateur> | Car <identificateur>

**🡺 La production est LL(1)**

10 ) <liste d'instructions> =:: 𝜀 | <instruction> ; <liste d'instructions>

**🡺 La production est LL(1) (Pr(**instruction**)ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

11 ) <instruction> =:: <expression>

|<identificateur> <instruction ‘>

| retour <expression>

| si <expression> alors { <liste d’instructions> } <instruction ‘’>

| tantque (<expression>) faire { < liste d’instructions > }

| ecrire( <expression> )

**🡺 La production n’est pas LL(1) (Pr(identificateur) ᴖ Pr(expression) = lettre )**

11 ) <instruction ‘> =:: =<instruction ‘’’> | [<expression>]= <instruction ’’’>

**🡺 La production est LL(1)**

11 ) <instruction ‘’’> = :: lire() | <expression>

**🡺 La production est LL(1) (PrᴖPr=Φ)**

11 ) <instruction ‘‘> =:: sinon { <liste d’instructions> } | 𝜀

**🡺 La production est LL(1)  (PrᴖSv=Φ)**

12 ) <expression> =:: <expression logique> <expression ‘>

**🡺 La production est LL(1)**

12 ) <expression ‘> =:: <operateur logique> <expression logique> <expression ‘> | 𝜀

**🡺 La production est LL(1) (PrᴖSv=Φ)**

13 ) <expression logique> =:: <expression simple><expression logique ‘>

**🡺 La production est LL(1)**

13 ) <expression logique ‘> =:: < comparaison> <expression simple> <expression logique ‘>| 𝜀

**🡺 La production est LL(1) (Pr(**compaison**)ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

14 ) <expression simple> =:: <terme><expression simple ‘> | -<terme><expression simple ‘>

**🡺 La production est LL(1) (Pr(**terme**)ᴖPr**(-)**=Φ )**

14 ) <expression simple ‘> =:: +<terme><expression simple ‘>

| -<terme><expression simple ‘>

| 𝜀

**🡺 La production est LL(1) ( ᴖPr = Φ et PrᴖSv=Φ)**

15 ) <terme> =:: <terme prioritaire> <terme ‘>

**🡺 La production est LL(1)**

15 ) <terme ‘> =:: \*<facteur><terme ‘>

| /<facteur><terme ‘>

| 𝜀

**🡺 La production est LL(1) ( ᴖPr** **= Φ et PrᴖSv=Φ)**

16 ) <terme prioritaire>=::!<facteur>|<facteur>

**🡺 La production est LL(1) (Pr(**facteur**)ᴖPr**(!)**=Φ et PrᴖSv=Φ)**

17 ) <facteur> =:: <identificateur> <facteur ‘>

| <cste>

| (<expression>)

| ’<lettre>’

**🡺 La production est LL(1) (ᴖPr =Φ)**

17 ) <facteur ‘> =:: 𝜀 | [ <expression> ] | (< paramètres effectifs >)

**🡺 La production est LL(1) ( ᴖPr** **= Φ et PrᴖSv=Φ)**

18 ) < paramètres effectifs > =:: 𝜀 | <expressions>

**🡺 La production est LL(1) (Pr(**expression**)ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

19 ) <expressions> =:: <expression> <expressions ‘>

**🡺 La production est LL(1)**

19 ) <expressions ‘> =:: , <expression> <expressions ‘>| 𝜀

**🡺 La production est LL(1) (Pr(** , **)ᴖPr**(𝜀)**=Φ et PrᴖSv=Φ)**

20 ) <operateur logique> =:: | | &

**🡺 La production est LL(1)**

21 ) <comparaison> =:: <|>| == | <= | >= | !=

**🡺 La production est LL(1)**

22 ) <identificateur> =:: <lettre> <mot>

**🡺 La production est LL(1)**

23 ) <mot> = :: 𝜀 | <lettre><mot> | <chiffre><mot>

**🡺 La production est LL(1) (ᴖPr =Φ et PrᴖSv=Φ)**

24 ) <cste> =:: <chiffre><cste ‘>

**🡺 La production est LL(1)**

24 ) <cste ‘> =:: 𝜀 | <cste>

**🡺 La production est LL(1) (ᴖPr =Φ et PrᴖSv=Φ)**

25 ) <chiffre> =:: 0|1|…|8|9

26 ) <lettre> =:: A|B|…|Z|a|b|…|z

# 5- Version finale de grammaire LL(1)

1 ) <programme> =:: <liste de déclarations> <liste de fonctions>

2 ) <liste de fonctions> =:: <fonction> <liste de fonctions> | main() { <liste d'instructions> }

3 ) <fonction>= ::< identificateur>(<liste de paramètres>)<liste de déclarations>{ <liste d'instructions> }

4 ) <liste de déclarations>= ::< déclarations> ; | 𝜀

5 ) < déclarations> =:: <déclaration> <déclarations ‘>

5 ) <déclarations ‘> = :: , <déclaration> <déclarations ‘> | 𝜀

6 ) <déclaration> =:: entier <déclaration ‘> | Car <déclaration ‘>

6 ) <déclaration ‘> =:: <identificateur><déclaration ‘’>

6 ) <déclaration ‘’> =:: 𝜀 | [<expression>]

7 ) <liste de paramètres>= :: 𝜀 | <paramètres>

8 ) < paramètres> =:: <paramètre> < paramètres ‘>

8 ) < paramètres ’> =:: , <paramètre> <paramètres ‘> | 𝜀

9 ) <paramètre> =:: entier <identificateur> | Car <identificateur>

10 ) <liste d'instructions> =:: 𝜀 | <instruction> ; <liste d'instructions>

11 ) <instruction> =:: <identificateur> <instruction ‘>

| retour <expression>

| si <expression> alors { <liste d’instructions> } <instruction ‘’>

| tantque (<expression>) faire { < liste d’instructions > }

| ecrire( <expression> )

11 ) <instruction ‘> =:: =<instruction ‘’’> | [<expression>]= <instruction ’’’>

11 ) <instruction ‘’’> = :: lire() | <expression>

11 ) <instruction ‘‘> =:: sinon { <liste d’instructions> } | 𝜀

12 ) <expression> =:: <expression logique> <expression ‘>

12 ) <expression ‘> =:: <operateur logique> <expression logique> <expression ‘> | 𝜀

13 ) <expression logique> =:: <expression simple><expression logique ‘>

13 ) <expression logique ‘> =:: < comparaison> <expression simple> <expression logique ‘>| 𝜀

14 ) <expression simple> =:: <terme><expression simple ‘> | -<terme><expression simple ‘>

14 ) <expression simple ‘> =:: +<terme><expression simple ‘>

| -<terme><expression simple ‘>

| 𝜀

15 ) <terme> =:: <terme prioritaire> <terme ‘>

15 ) <terme ‘> =:: \*<facteur><terme ‘>

| /<facteur><terme ‘>

| 𝜀

16 ) <terme prioritaire>=::!<facteur>|<facteur>

17 ) <facteur> =:: <identificateur> <facteur ‘>

| <cste>

| (<expression>)

| ’<lettre>’

17 ) <facteur ‘> =:: 𝜀 | [ <expression> ] | (< paramètres effectifs >)

18 ) < paramètres effectifs > =:: 𝜀 | <expressions>

19 ) <expressions> =:: <expression> <expressions ‘>

19 ) <expressions ‘> =:: , <expression> <expressions ‘>| 𝜀

20 ) <operateur logique> =:: | | &

21 ) <comparaison> =:: <|>| == | <= | >= | !=

22 ) <identificateur> =:: <lettre> <mot>

23 ) <mot> = :: 𝜀 | <lettre><mot> | <chiffre><mot>

24 ) <cste> =:: <chiffre><cste ‘>

24 ) <cste ‘> =:: 𝜀 | <cste>

25 ) <chiffre> =:: 0|1|…|8|9

26 ) <lettre> =:: A|B|…|Z|a|b|…|z

# 6- Schéma de traduction :

1 ) <programme> =:: <liste de déclarations> <liste de fonctions>

2 ) <liste de fonctions> =:: <fonction> <liste de fonctions> | main() { <liste d'instructions> }

3 ) <fonction>= ::< identificateur>(<liste de paramètres>)<liste de déclarations>{ <liste d'instructions> }

4 ) <liste de déclarations>= :: {creationTS} < déclarations> ; | 𝜀

5 ) < déclarations> =:: <déclaration> <déclarations ‘>

5 ) <déclarations ‘> = :: , <déclaration> <déclarations ‘> | 𝜀

6 ) <déclaration> =:: entier <déclaration ‘> {ajouterIdent( ident, entier)} | Car <déclaration ‘> {ajouterIdent( ident, car)}

6 ) <déclaration ‘> =:: <identificateur> {si estDejaDeclare(variable.attr) => err} <déclaration ‘’>

6 ) <déclaration ‘’> =:: 𝜀 | [<expression>]

7 ) <liste de paramètres>= :: 𝜀 | <paramètres>

8 ) < paramètres> =:: <paramètre> < paramètres ‘>

8 ) < paramètres ’> =:: , <paramètre> <paramètres ‘> | 𝜀

9 ) <paramètre> =:: entier <identificateur> {ajouterIdent( ident, entier)} | Car <identificateur> {ajouterIdent( ident, car)}

10 ) <liste d'instructions> =:: 𝜀 | <instruction> ; <liste d'instructions>

11 ) <instruction> =:: <identificateur> {si !estDejaDeclare(variable.attr) => err , IP.he = var IP.type=var.type} <instruction ‘>

| retour <expression>

| si <expression> alors { <liste d’instructions> } <instruction ‘’>

| tantque (<expression>) faire { < liste d’instructions > }

| ecrire( <expression> )

11 ) <instruction ‘> =:: ={IT.type = var.type}<instruction ‘’’> | [<expression>]={IS.he = IP.he } <instruction ’’’>

11 ) <instruction ‘’’> = :: lire() | <expression>

11 ) <instruction ‘‘> =:: sinon { <liste d’instructions> } | 𝜀

12 ) <expression> =:: {EXPL.type = EXP.type}<expression logique> <expression ‘>

12 ) <expression ‘> =:: <operateur logique> <expression logique> <expression ‘> | 𝜀

13 ) <expression logique> =:: {EXPSI.type = EXPL.type}<expression simple><expression logique ‘>

13 ) <expression logique ‘> =:: < comparaison> <expression simple> <expression logique ‘>| 𝜀

14 ) <expression simple> =:: {EXPSI.type = T.type}<terme><expression simple ‘> | -<terme><expression simple ‘>

14 ) <expression simple ‘> =:: +<terme><expression simple ‘>

| -<terme><expression simple ‘>

| 𝜀

15 ) <terme> =:: {TPrio.type = T.type}<terme prioritaire> <terme ‘>

15 ) <terme ‘> =:: \*<facteur><terme ‘>

| /<facteur><terme ‘>

| 𝜀

16 ) <terme prioritaire>=:: !<facteur>| {Fac.type = TPrio.type}<facteur>

17 ) <facteur> =:: <identificateur> {si !estDejaDeclare(variable.attr) => err } <facteur ‘>

| <cste>

| (<expression>)

| ’<lettre>’

17 ) <facteur ‘> =:: 𝜀 | [ <expression> ] | (< paramètres effectifs >)

18 ) < paramètres effectifs > =:: 𝜀 | <expressions>

19 ) <expressions> =:: <expression> <expressions ‘>

19 ) <expressions ‘> =:: , <expression> <expressions ‘>| 𝜀

20 ) <operateur logique> =:: | | &

21 ) <comparaison> =:: <|>| == | <= | >= | !=

22 ) <identificateur> =:: <lettre> <mot>

23 ) <mot> = :: 𝜀 | <lettre><mot> | <chiffre><mot>

24 ) <cste> =:: <chiffre><cste ‘>

24 ) <cste ‘> =:: 𝜀 | <cste>

25 ) <chiffre> =:: 0|1|…|8|9

26 ) <lettre> =:: A|B|…|Z|a|b|…|z