Projet Théorie de compilation

2GI

Projet C-Pascal

Réalisé par :

- OUTIDRARINE Mohamed
- ECH-CHEBLAOUI Yassine
- ZAHAD Zakaria
-AMGHAR Souhail

Encadrant :

- Pr. Souhail GHAZI

Sommaire:

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 , X→b

Alors :

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**)∩Pr**(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**)∩Pr**(ε)**=Φ et Pr∩Sv=Φ)**

5) < déclarations> =:: <déclaration> <déclarations ‘>
➔ **La production est LL(1)**

5) <déclarations ‘> = :: , <déclaration> <déclarations ‘> | ε
➔ **La production est LL(1) (Pr(,)∩Pr**(ε)**=Φ et Pr∩Sv=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(** [**)∩Pr**(ε)**=Φ et Pr∩Sv=Φ)**

7) <liste de paramètres>= :: ε | <paramètres>
➔ **La production est LL(1) (Pr(**paramètres**)∩Pr**(ε)**=Φ et Pr∩Sv=Φ)**

8) < paramètres> =:: <paramètre> < paramètres ‘>
➔ **La production est LL(1)**

8) < paramètres ’> =:: , <paramètre> <paramètres ‘> | ε
➔ **La production est LL(1) (Pr(,)∩Pr**(ε)**=Φ et Pr∩Sv=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**)∩Pr(**ε**)=Φ et Pr∩Sv=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) ∩ Pr(expression) = lettre)**

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

➔ **La production est LL(1)**

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

➔ **La production est LL(1) (Pr∩Pr=Φ)**

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

➔ **La production est LL(1) (Pr∩Sv=Φ)**

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

➔ **La production est LL(1)**

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

➔ **La production est LL(1) (Pr∩Sv=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**)∩Pr(**ε**)=Φ et Pr∩Sv=Φ)**

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

➔ **La production est LL(1) (Pr(**terme**)∩Pr(**-**)=Φ)**

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

➔ **La production est LL(1) (∩Pr = Φ et Pr∩Sv=Φ)**

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

➔ **La production est LL(1)**

15) <terme ‘> =:: *<facteur><terme ‘>
| /<facteur><terme ‘>
| ε
➔ **La production est LL(1) (∩Pr = Φ et Pr∩Sv=Φ)**

16) <terme prioritaire>=::!<facteur>|<facteur>
➔ **La production est LL(1) (Pr(**facteur**)∩Pr**(!)**=Φ et Pr∩Sv=Φ)**

17) <facteur> =:: <identificateur> <facteur ‘>
| <cste>
| (<expression>)
| ’<lettre>’
➔ **La production est LL(1) (∩Pr =Φ)**

17) <facteur ‘> =:: ε | [<expression>] | (< paramètres effectifs >)
➔ **La production est LL(1) (∩Pr = Φ et Pr∩Sv=Φ)**

18) < paramètres effectifs > =:: ε | <expressions>
➔ **La production est LL(1) (Pr(**expression**)∩Pr**(ε)**=Φ et Pr∩Sv=Φ)**

19) <expressions> =:: <expression> <expressions ‘>
➔ **La production est LL(1)**

19) <expressions ‘> =:: , <expression> <expressions ‘>| ε
➔ **La production est LL(1) (Pr(,)∩Pr**(ε)**=Φ et Pr∩Sv=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) (∩Pr =Φ et Pr∩Sv=Φ)**

24) <cste> =:: <chiffre><cste ‘>
➔ **La production est LL(1)**

24) <cste ‘> =:: ε | <cste>
➔ **La production est LL(1) (∩Pr =Φ et Pr∩Sv=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 :

INTRODUCTION :

Certaines expressions peuvent être syntaxiquement correctes mais ne pas avoir de sens par rapport à la sémantique du langage. Par exemple, en C, int a=4.12 est correct syntaxiquement mais n'a pas de sémantique. L'analyse sémantique étudie l'arbre de syntaxe abstraite produit par l'analyse syntaxique pour éliminer au maximum les programmes qui ne sont pas corrects du point de vue de la sémantique.

Ainsi, Après l'analyse lexicale et l'analyse syntaxique, l'étape suivante dans la conception d'un compilateur est l'analyse sémantique dont la partie la plus visible est le contrôle de type dont on traitera les tâches suivantes :

- construire et mémoriser des représentations des types définis par l'utilisateur autant que structures.
- traiter les déclarations des variables et mémoriser leurs types.
- vérifier que toute variable référencée est préalablement déclarée.
- contrôler les types des opérandes des opérations arithmétiques.
- contrôler les types des opérandes des opérations logiques.
- Contrôler les types des opérandes de l'affichage et de l'écriture.
- Etc...

ETUDE DE LA GRAMMAIRE :

- Création d'une structure Variable pour stocker les identificateurs et leurs types et la taille si l'identificateur est de type tableau :

```
typedef struct
{  string val;
   string type; int nb=0;
   bool estfct;
   vector<variable> param ;
   int local_global;
   } Variable;
```

- ajouterTS(variable) ajoute la variable à la table des symboles et verifie si il existe déjà ; si c'est le cas elle renvoie une erreur
- typeTS(identif.val, TS) retourne le type de la valeur de l'identifiant à partir de TS ; sinon retourne erreur

- verifierTableau(identif.val) verifie si le identif.val est un tableau ; sinon retourne erreur
- verifierFonction(identif.val,tableautype) verifie si le identif.val est une fonction bien utilisée y compris ses paramètres ; sinon retourne erreur

- La Déclaration :

1) <programme> =:: {creationTS} <liste de déclarations> <liste de fonctions>

3) <fonction>= ::< identificateur>(<liste de paramètres>)
{variable.val=identif.valex, variable.estfct=true , variable.param=listepara.val[] ,ajouterTS(variable)}
<liste de déclarations>{ <liste d'instructions> }

6) <déclaration> =:: entier {variable.Type='entier'} <déclaration '>| Car {variable.Type= 'car'}
<déclaration '>

6) <déclaration '> =:: <identificateur> {variable.val=identif.valex ,ajouterTS(variable)} <déclaration ''>

6) <déclaration ''> =:: ε | [<expression>] {variable.nb=le nombre de paramètre}

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

8) < paramètres> =:: <paramètre> listepara.val[].ajouter(para.val)| < paramètres>,<paramètre>
listepara.val[].ajouter(para.val)

9) <paramètre> =:: entier {variable.Type='entier', para.val=var.type} <identificateur>
{variable.val=identif.valex ,ajouterTS(variable)}
| Car {variable.Type='car', para.val=var.type } <identificateur>
{variable.val=identif.valex ,ajouterTS(variable)}

- Compatibilité entre les types:

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

11) <instruction> =:: <identificateur> {instru'.val= identif.type ; tmp =identif.val } <instruction '>
| retour <expression>
| si <expression> alors { <liste d'instructions> } <instruction ''>
| tantque (<expression>) faire { < liste d'instructions > }
| ecrire (<expression>)

11) <instruction '> =:: = <instruction '''>
| [{verifierTableau(tmp)} <expression>]= <instruction '''>
{si instru'.val != instru'''.val alors erreur « conflicting type »}

11) <instruction ‘’’> = :: lire() | <expression> { instru’’’.val=expr.val}

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 ‘>
{si term.val != expr’.val alors erreur « conflicting type »}
{expr.val=term.val}

14) <expression simple ‘> =:: +<terme><expression simple ‘> {expr’.val=term.val} {si term.val != ‘entier’ alors erreur « conflicting type »}

| -<terme><expression simple ‘> {expr’.val=term.val} {si term.val != ‘entier’ alors erreur « conflicting type »}

| ε

15) <terme> =:: <terme prioritaire><terme ‘>
{si term’.val != termprio.val alors erreur « conflicting type »}
{term.val=termprio.val}

15) <terme ‘> =:: *<facteur><terme ‘> {term’.val=fac.val} {si term’.val != ‘entier’ alors erreur « conflicting type »}

| /<facteur><terme ‘> {term’.val=fac.val} {si term’.val != ‘entier’ alors erreur « conflicting type »}

| ε

16) <terme prioritaire>=:: !<facteur>| <facteur> {termprio.val=fac.val}

17) <facteur> =:: <identificateur> {fac.val= identif.type ; tmp=identif.val} <facteur ‘>
| <cste> {fac.val=‘entier’}
| (<expression>)
| ’<lettre>’ {fac.val=‘car’}

17) <facteur ‘> =:: ε
| [{verifierTableau(tmp)} <expression>]
| (< paramètres effectifs >) {verifierFonction(tmp , paraeff.val[])}

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

19) <expressions> =:: <expression> <expressions ‘> { paraeff.val[].ajouter(expr.val) }

19) <expressions ‘> =:: , <expression> <expressions ‘> { paraeff.val[].ajouter(expr.val) }| 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