Projet Théorie de compilation

2GI

Projet C-Pascal

Réalisé par :

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

Encadrant :

- Pr. Malika ADDOU

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 déclarations> =:: ε | <déclaration> , <liste de déclarations> ;

3 <déclaration> =:: entier <identificateur> | Car <identificateur>
| entier <identificateur> [<cste>]
| Car <identificateur> [<cste>]

4 <liste de fonctions> =:: ε | <déclaration fonction> <liste d'instructions fonctions> <liste de fonctions>

5 <déclaration fonction> =:: <identificateur> (<liste de paramètres>)

6 <liste de paramètres> =:: ε | <paramètre> , <liste de paramètres>

7 <paramètre> =:: entier <identificateur> | Car <identificateur>

8 <liste d'instructions fonctions> =:: <liste de déclarations> { <liste d'instructions> }

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

10 <instruction> =:: <expression>

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

11 <expression> =:: <expression simple> <operateur logique> <expression simple>
| <expression simple>

12 <liste d'expressions> =:: ε | <expression> , <liste d'expressions>

13 <expression simple> =:: <expression simple>+<terme>
| <expression simple>-<terme>
| <expression simple><comparaison><terme>
| <terme>
| -<terme>

14 <terme> =:: <terme>*<facteur>
| <terme>/<facteur>
| !<facteur>
| <facteur>

15 <facteur> =:: <identificateur>
| <identificateur>(<liste d'expressions>)
| <cste>
| (<expression simple>)
| <identificateur>[<expression simple>]
| <lettre>

16 <operateur logique> =:: | | &

17 <comparaison> =:: <|>| == | <= | >= | !=

18 <identificateur> =:: <lettre> <mot>

19 <mot> = :: ε | <lettre><mot> | <chiffre><mot>

20 <cste> =:: <chiffre> | <chiffre><cste>

21 <chiffre> =:: 0|1|…|8|9

22 <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 :

```
13 <expression simple>  =::      <expression simple>+<terme>
                                 | <expression simple>-<terme>
                                 | <expression simple><comparaison><terme>
                                 | <terme>
                                 | -<terme>

14 <terme>  =:: <terme>*<facteur>
                | <terme>/<facteur>
                | !<facteur>
                | <facteur>
```

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

```
13 <expression simple> =::   <terme><expression simple '> | -<terme><expression simple '>

13' <expression simple '> =::  +<terme><expression simple '>
                               | -<terme><expression simple '>
                               | <comparaison><terme><expression simple '>
                               | ε

14 <terme> =::   !<facteur><terme '> | <facteur><terme '>

14' <terme '> =::          *<facteur><terme '>
                           | /<facteur><terme '>
                           | ε
```

- Factorisation de la grammaire :

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

```
3 <déclaration>  =::  entier <identificateur> | Car <identificateur>
                     | entier <identificateur> [<cste>]
                     | Car <identificateur> [<cste>]



10 <instruction> =::    <expression>
                     |<identificateur>=<expression>
                     |<identificateur>[<expression simple>] =<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()

11 <expression> =::     <expression simple> <operateur logique> <expression simple>
                     | <expression simple>


15 <facteur> =::        <identificateur>
                     | <identificateur>(<liste d'expressions>)
                     | <cste>
                     | (<expression simple>)
                     | <identificateur>[<expression simple>]
                     | <lettre>

20 <cste>  =:: <chiffre> | <chiffre><cste>
```

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

```
3 <déclaration>  =::  entier <déclaration '> | Car <déclaration '>

3'<déclaration '> =::  <identificateur><déclaration ''>

3''<déclaration ''> =:: ε  | [<cste>]


10 <instruction> =::    <expression>
```

|<identificateur><instruction ‘>
| retour <expression>
| si <expression> alors { <liste d’instructions> } <instruction ‘’>
| tantque <expression> faire { <liste d’instructions> }
| ecrire(<expression>)

10’ <instruction ‘> =:: =lire()
| [<expression simple>] =<expression>
| =<expression>

10’’ <instruction ‘’> =:: sinon { <liste d’instruction> } | ε

11 <expression> =:: <expression simple> <expression ’>

11’ <expression ’> =:: <operateur logique > <expression simple> | ε

15 <facteur> =:: <identificateur><facteur ‘>
| <cste>
| (<expression simple>)
| <lettre>

15’ <facteur ‘> =:: ε | [<expression simple>] | (<liste d’expressions>)

20 <cste> =:: <chiffre><cste ‘>

20’ <cste ‘> =:: ε | <cste>

- Grammaire Après Changement :

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

2 <liste de déclarations> =:: ε | <déclaration> , <liste de déclarations> ;

3 <déclaration> =:: entier <déclaration ‘> | Car <déclaration ‘>

3’<déclaration ‘> =:: <identificateur><déclaration ‘’>

3’’<déclaration ‘’> =:: ε | [<cste>]

4 <liste de fonctions> =:: ε | <déclaration fonction> <liste d'instructions fonctions> <liste de fonctions>

5 <déclaration fonction> =:: <identificateur> (<liste de paramètres>)

6 <liste de paramètres> =:: ε | <paramètre> , <liste de paramètres>

7 <paramètre> =:: entier <identificateur> | Car <identificateur>

8 <liste d'instructions fonctions> =:: <liste de déclarations> { <liste d'instructions> }

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

10 <instruction> =:: <expression>
| <identificateur><instruction ‘>
| retour <expression>
| si <expression> alors { <liste d’instructions> } <instruction ‘’>
| tantque <expression> faire { <liste d’instructions> }
| ecrire(<expression>)

10’ <instruction ‘> =:: =lire()
| [<expression simple>] =<expression>
| =<expression>

10’’ <instruction ‘’> =:: sinon { <liste d’instruction> } | ε

11 <expression> =:: <expression simple> <expression ’>

11’ <expression ’> =:: <operateur logique > <expression simple> | ε

12 <expression simple> =:: <terme><expression simple ‘> | -<terme><expression simple ‘>

12’ <expression simple ‘> =:: +<terme><expression simple ‘>
| -<terme><expression simple ‘>
| <comparaison><terme><expression simple ‘>
| ε

13 <terme> =:: !<facteur><terme ‘> | <facteur><terme ‘>

13’ <terme ‘> =:: *<facteur><terme ‘>
| /<facteur><terme ‘>
| ε

14 <facteur> =:: <identificateur><facteur ‘>
| <cste>
| (<expression simple>)
| <lettre>

14’ <facteur ‘> =:: ε | [<expression simple>] | (<liste d’expressions>)

15 <liste d’expressions> =:: ε | <expression> , <liste d’expressions>

16 <operateur logique> =:: | | &

17 <comparaison> =:: <|>| == | <= | >= | !=

18 <identificateur> =:: <lettre> <mot>

19 <mot> = :: ε | <lettre><mot> | <chiffre><mot>

20 <cste> =:: <chiffre><cste '>

20' <cste '> =:: ε | <cste>

21 <chiffre> =:: 0|1|…|8|9

22 <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 déclarations >	Entier, car , ε	; , lettre, $
<déclaration >	entier, car	,
<déclaration '>	Lettre, ε	,
<déclaration ''>	[, ε	,
<liste de fonctions >	lettre , ε	$
<déclaration fonction>	lettre	entier, car , lettre , $
<liste de paramètres>	Entier, car , ε	)
<paramètre >	Entier, car	,
<liste d'instructions fonctions >	entier, car, ε	lettre, $

<liste d'instructions>	Retour, si, tantque, ecrire, ε	}
<instruction>	Retour, si, tantque, ecrire, lettre , chiffre, (, !, -	}, ;
<instruction ‘>	=, [	}, ;
<instruction ‘’>	sinon, ε	}, ;
<expression>	Lettre, chiffre, (, !, -	, ,alors, faire, }, ; ,)
<expression ‘>	&, | , ε	, ,alors, faire, }, ; ,)
<liste d'expressions>	Lettre, chiffre, (, !, -, ε	)
<expression simple>	Lettre, chiffre, (, ! , -	),], &, | , , ,alors, faire, }, ;
<expression simple ‘>	+,-,<, >, ==, <=, >=, !=, ε	),], &, | , , ,alors, faire, }, ;
<terme>	! , Lettre, chiffre, (	<, >, ==, <=, >=, != ,),], &, | , , ,alors, faire, }, ;
<terme ‘>	*,/ , ε	<, >, ==, <=, >=, != ,),], &, | , , ,alors, faire, }, ;

<facteur>	Lettre, Chiffre, (	*,/ , <, >, ==, <=, >=, != ,),], &, | , , ,alors, faire, }, ;
<facteur '>	[, ε , (	*,/ , <, >, ==, <=, >=, != ,),], &, | , , ,alors, faire, }, ;
<comparaison>	<, >, ==, <=, >=, !=	! , Lettre, chiffre, (
<operateur logique >	|, &	Lettre, chiffre, (, ! , -
<cste>	chiffre	*,/ , <, >, ==, <=, >=, != ,),], &, | , , ,alors, faire, }, ;
<cste '>	chiffre , ε	*,/ , <, >, ==, <=, >=, != ,),], &, | , , ,alors, faire, }, ;
<identificateur>	lettre	*,/ , <, >, ==, <=, >=, != ,),], &, | , , ,alors, faire, }, ; , (, =, [
<mot>	Lettre, chiffre , ε	*,/ , <, >, ==, <=, >=, != ,),], &, | , , ,alors, faire, }, ; , (, =, [

Vérification des productions :

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

➔ **La production est LL(1)**

2 <liste de déclarations> =:: ε | <déclaration> , <liste de déclarations> ;

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

3 <déclaration> =:: entier <déclaration '> | Car <déclaration '>

➔ **La production est LL(1)**

3'<déclaration '> =:: <identificateur><déclaration ''>

➔ **La production est LL(1)**

3''<déclaration ''> =:: ε | [<cste>]

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

4 <liste de fonctions> =:: ε | <déclaration fonction> <liste d'instructions fonctions> <liste de fonctions>

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

5 <déclaration fonction> =:: <identificateur> (<liste de paramètres>)

➔ **La production est LL(1)**

6 <liste de paramètres> =:: ε | <paramètre> , <liste de paramètres>

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

7 <paramètre> =:: entier <identificateur> | Car <identificateur>

➔ **La production est LL(1)**

8 <liste d'instructions fonctions> =:: <liste de déclarations> { <liste d'instructions> }

➔ **La production est LL(1)**

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

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

10 <instruction> =:: <identificateur><instructions ‘>
| retour <expression>
| <expression>
| si <expression> alors { <liste d’instructions> } <instruction ‘’>
| tantque <expression> faire { <liste d’instructions> }
| ecrire(<expression>)

➔ **La production n’est pas LL(1) (∩Pr !=Φ)**

10’ <instruction ‘> =:: =lire()
| [<expression simple>] =<expression>
| =<expression>

➔ **La production est LL(1)**

10’’ <instruction ‘’> =:: sinon { <instruction> } | ε

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

11 <expression> =:: <expression simple> <expression ’>

➔ **La production est LL(1)**

11’ <expression ’> =:: <operateur logique > <expression simple> | ε

➔ **La production est LL(1)**

12 <expression simple> =:: <terme><expression simple ‘> | -<terme><expression simple ‘>

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

12’ <expression simple ‘> =:: +<terme><expression simple ‘>
| -<terme><expression simple ‘>

| <comparaison><terme><expression simple ‘>
| ε

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

13 <terme> =:: !<facteur><terme ‘> | <facteur><terme ‘>

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

13’ <terme ‘> =:: *<facteur><terme ‘>
| /<facteur><terme ‘>
| ε

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

14 <facteur> =:: <identificateur><facteur ‘>
| <cste>
| (<expression simple>)
| <lettre>

➔ **La production n’est pas LL(1) (∩Pr !=Φ)**

14’ <facteur ‘> =:: ε
| [<expression simple>]
| (<liste d’expressions>)

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

15 <liste d’expressions> =:: ε | <expression> , <liste d’expressions>

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

16 <operateur logique> =:: | | &

➔ **La production est LL(1)**

17 <comparaison> =:: <|>| == | <= | >= | !=

➔ **La production est LL(1)**

18 <identificateur> =:: <lettre> <mot>

➔ **La production est LL(1)**

19 <mot> = :: ε | <lettre><mot> | <chiffre><mot>

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

20 <cste> =:: <chiffre><cste ‘>

➔ **La production est LL(1)**

20’ <cste ‘> =:: ε | <cste>

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

5- Version finale de grammaire LL(1)

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

2 <liste de déclarations> =:: ε | <déclaration> , <liste de déclarations> ;

3 <déclaration> =:: entier <déclaration ‘> | Car <déclaration ‘>

3’<déclaration ‘> =:: <identificateur><déclaration ‘’>

3’’<déclaration ‘’> =:: ε | [<cste>]

4 <liste de fonctions> =:: ε | <déclaration fonction> <liste d'instructions fonctions> <liste de fonctions>

5 <déclaration fonction> =:: <identificateur> (<liste de paramètres>)

6 <liste de paramètres> =:: ε | <paramètre> , <liste de paramètres>

7 <paramètre> =:: entier <identificateur> | Car <identificateur>

8 <liste d'instructions fonctions> =:: <liste de déclarations> { <liste d'instructions> }

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

10 <instruction> =:: <expression>
|<identificateur><instruction ‘>
| retour <expression>
| si <expression> alors { <liste d’instructions> } <instruction ‘’>
| tantque <expression> faire { <liste d’instructions> }
| ecrire(<expression>)

10’ <instruction ‘> =:: =lire()
| [<expression simple>] =<expression>
| =<expression>

10’’ <instruction ‘’> =:: sinon { <liste d’instruction> } | ε

11 <expression> =:: <expression simple> <expression ’>

11’ <expression ’> =:: <operateur logique > <expression simple> | ε

12 <expression simple> =:: <terme><expression simple ‘> | -<terme><expression simple ‘>

12’ <expression simple ‘> =:: +<terme><expression simple ‘>

| -<terme><expression simple ‘>
| <comparaison><terme><expression simple ‘>
| ε

13 <terme> =:: !<facteur><terme ‘> | <facteur><terme ‘>

13’ <terme ‘> =:: *<facteur><terme ‘>
| /<facteur><terme ‘>
| ε

14 <facteur> =:: <identificateur><facteur ‘>
| <cste>
| (<expression simple>)
| <lettre>

14’ <facteur ‘> =:: ε | [<expression simple>] | (<liste d’expressions>)

15 <liste d’expressions> =:: ε | <expression> , <liste d’expressions>

16 <operateur logique> =:: | | &

17 <comparaison> =:: <|>| == | <= | >= | !=

18 <identificateur> =:: <lettre> <mot>

19 <mot> = :: ε | <lettre><mot> | <chiffre><mot>

20 <cste> =:: <chiffre><cste ‘>

20’ <cste ‘> =:: ε | <cste>

21 <chiffre> =:: 0|1|…|8|9

22 <lettre> =:: A|B|…|Z|a|b|…|z