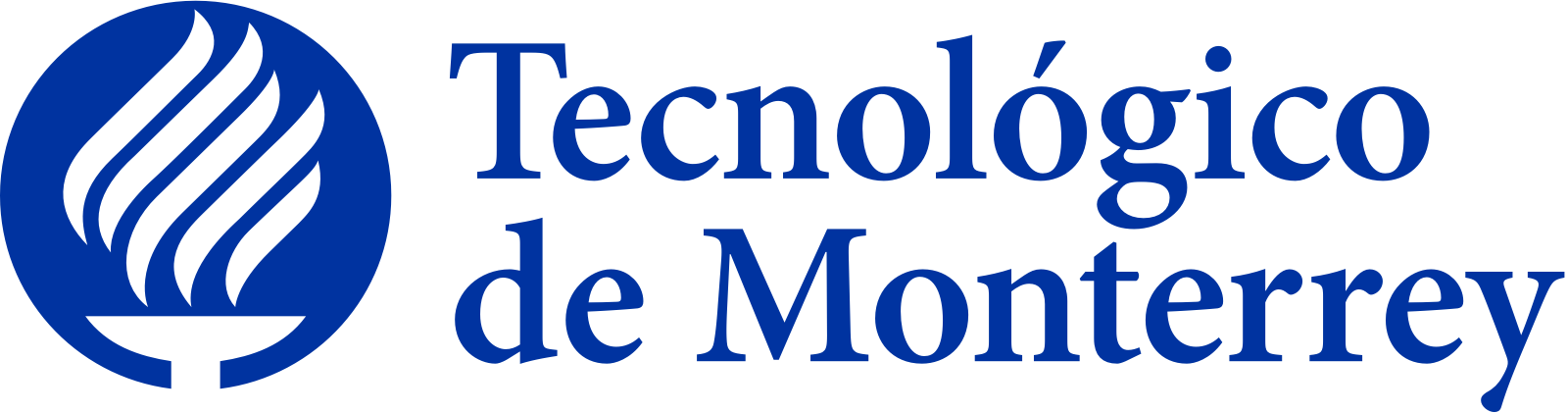
**Instituto Tecnológico de Monterrey**



Project II: Parser

**Campus: Querétaro**

**Adrián Torres Hernández**

**A01173530**

**Diseño de compiladores**

# 1.- Introducción

El analizador sintáctico o parser es una de las siguientes fases para crear un compilador. Su función principal es analizar una cadena de instrucciones escrita en un lenguaje de programación, descomponiéndola en componentes sintácticos lógicos. Este proceso es de suma importancia para la correcta traducción del código fuente a un lenguaje ejecutable. Se encarga de transformar las entradas en un formato utilizable para su posterior procesamiento, desglosando la estructura gramatical del código en sus elementos individuales. De esta manera, contribuye a garantizar que el código sea traducido de manera precisa y eficiente.

## 1.1 Resumen

El presente reporte presenta los resultados obtenidos en las fases de desarrollo de software para la segunda parte en la creación de un compilador "Parser" el cual es el analizador sintáctico. El reporte cuenta con la fase de análisis, la cual se encarga en dar una explicación de los componentes del lenguaje Pascal, el desarrollo del Autómata Finito Determinista con la tabla de transiciones y tabla de Tokens ID. Posterior a la fase de análisis sigue la de diseño que se encarga de dar una aproximación a la implementación con pseudo código de como implementar la parte de análisis. Después sigue la fase de implementación en donde se dará una explicación de lo que se hizo en el código. Por último, se termina con la fase de validación y verificación en donde se mostrarán los casos de pruebas con su respectiva entrada y salida.

## 1.2.- Notación

ÍNDICE

[1.- Introducción 2](#_Toc152148077)

[1.1 Resumen 2](#_Toc152148078)

[1.2.- Notación 2](#_Toc152148079)

[2.- Análisis 3](#_Toc152148080)

[2.1.- Quitar ambigüedad 5](#_Toc152148081)

[Incongruencias en la gramática 5](#_Toc152148082)

[Regla de precedencia 5](#_Toc152148083)

[2.2.- Recursividad por la izquierda 7](#_Toc152148084)

[2.3.- Factorización a la izquierda 12](#_Toc152148085)

[3.- Diseño 15](#_Toc152148086)

[3.1.- Gramática final 15](#_Toc152148087)

[3.2.- Calculo de los First 17](#_Toc152148088)

[3.3.- Calculo de los Follow 19](#_Toc152148089)

[3.4.- Calculo de los First+ 23](#_Toc152148090)

[4.- Implementación 26](#_Toc152148091)

[5.- Verificación y Validación 27](#_Toc152148092)

[6.- Referencias 27](#_Toc152148093)

# 2.- Análisis

La gramática asignada en un inicio fue la siguiente:

1. start → **program ID** **;** declaration\_block main\_block

2. declaration\_block → vars\_block functions\_block procedures\_block

3. main\_block → compound\_statement **•**

4. vars\_block → **var** var\_declaration | ε

5. var\_declaration → var\_declaration var\_list **:** type\_specifier **;** | var\_list **:** type\_specifier **;**

6. var\_list → var\_list **,** **ID** | **ID**

7. type\_specifier → basic\_type | array\_type

8. basic\_type → **integer** | **real** | **string**

9. array\_type → **array [ NUMBER • • NUMBER ]** **of** basic\_type

10. functions\_block → functions\_block function\_declaration | ε

11. function\_declaration → **function ID(** params **)** **:** type\_specifier **;** local\_declarations compound\_stmt **;**

12. procedures\_block → procedures\_block procedure\_declaration | ε

13. procedure\_declaration → **procedure ID(** params **) ;** local\_declarations compound\_stmt **;**

14. params → param\_list | ε

15. param\_list → param\_list var\_list **:** type\_specifier **;** | var\_list **:** type\_specifier **;**

16. local\_declarations → vars\_block | ε

17. compound\_stmt → **begin** statement\_list **end**

18. statement\_list → statement\_list statement **;** | statement **;**

19. statement → assignment\_stmt | call \_stmt | compound\_stmt | selection\_stmt | for\_stmt | repeat\_stmt | input\_stmt | output\_stmt

20. assignment\_stmt → var **:=** arithmetic\_expression | var **:= STRING**

21. call\_stmt → call

22. selection\_stmt → **if** **(** logic\_expression **)** then statement **;** | **if** **(** logic\_expression **)** then statement else statement **;**

23. repeat\_stmt → **repeat** statement\_list **until** **(** logic\_expression **)**

24. for\_stmt → **for ID := NUMBER to NUMBER do** statement **;**

25. input\_stmt → **readln** **(** var\_list **)**

26. output\_stmt → **writeln** **(** output\_list **)**

27. output\_list → output\_list **,** output | output

28. output → arithmetic\_expression | **STRING**

29. var → **ID** | **ID** **[** arithmetic\_expression **]**

30. logic\_expression → arithmetic\_expression relop arithmetic\_expression

31. relop → **<=** | **<** | **>** | **>=** | **==** | **!=**

32. arithmetic \_expression → arithmetic\_expression arithmetic\_operator arithmetic\_expression | **(** arithmetic\_expression **)** | var | call | **NUMBER**

33. arithmetic\_operator → **+** | **-** | **\*** | **/**

34. call → **ID (** args **)**

35. args → arg\_list | ε

36. arg\_list → arg\_list **,** arithmetic\_expression | arithmetic\_expression

## 2.1.- Quitar ambigüedad

### Incongruencias en la gramática

En la regla 9, 24 y 32 tienen el símbolo terminal “NUMBER” que para mí implementación es ambigua porque separo los número enteros y reales en dos tablas diferentes por lo que se tiene que especificar si es un entero o un real y quedaría de la siguiente forma:

9. array\_type → **array [ int\_number • • int\_number ]** **of** basic\_type

24. for\_stmt → **for ID := int\_number to int\_number do** statement

34. factor → **(** arithmetic\_expression **)** | var | call | **int\_number**  | **real\_number**

Eliminamos punto y coma en las siguientes sentencias en nuestra gramática debido a que statement\_list ya contiene punto y coma haciéndolo innecesario para las siguientes sentencias

22. selection\_stmt → **if** **(** logic\_expression **)** then statement | **if** **(** logic\_expression **)** then statement else statement

23. repeat\_stmt → **repeat** statement\_list **until** **(** logic\_expression **)**

24. for\_stmt → **for ID := int\_number to int\_number do** statement

Si queremos que nuestro Writeln no truene al momento de estar vació entonces hay que hacer el siguiente cambio.

28. output → arithmetic\_expression | **STRING |** ε

### Regla de precedencia

Para las expresiones aritméticas no hay prioridad en los operadores por lo que hay que quitar la ambigüedad así eliminando la regla número 33 con:

arithmetic\_expression → term | term **+** term | term **-** term

term → factor | factor **\*** factor | factor **/** factor

factor → **(** arithmetic\_expression **)** | var | call | **int\_number**  | **real\_number**

Después de quitar ambigüedad la gramática es la siguiente:

1. start → **program ID** **;** declaration\_block main\_block

2. declaration\_block → vars\_block functions\_block procedures\_block

3. main\_block → compound\_statement **•**

4. vars\_block → **var** var\_declaration | ε

5. var\_declaration → var\_declaration var\_list **:** type\_specifier **;** | var\_list **:** type\_specifier **;**

6. var\_list → var\_list **,** **ID** | **ID**

7. type\_specifier → basic\_type | array\_type

8. basic\_type → **integer** | **real** | **string**

9. array\_type → **array [ int\_number • • int\_number ]** **of** basic\_type

10. functions\_block → functions\_block function\_declaration | ε

11. function\_declaration → **function ID(** params **)** **:** type\_specifier **;** local\_declarations compound\_stmt **;**

12. procedures\_block → procedures\_block procedure\_declaration | ε

13. procedure\_declaration → **procedure ID(** params **) ;** local\_declarations compound\_stmt **;**

14. params → param\_list | ε

15. param\_list → param\_list var\_list **:** type\_specifier **;** | var\_list **:** type\_specifier **;**

16. local\_declarations → vars\_block | ε

17. compound\_stmt → **begin** statement\_list **end**

18. statement\_list → statement\_list statement **;** | statement **;**

19. statement → assignment\_stmt | call \_stmt | compound\_stmt | selection\_stmt | for\_stmt | repeat\_stmt | input\_stmt | output\_stmt

20. assignment\_stmt → var **:=** arithmetic\_expression | var **:= STRING**

21. call\_stmt → call

22. selection\_stmt → **if** **(** logic\_expression **)** then statement | **if** **(** logic\_expression **)** then statement else statement

23. repeat\_stmt → **repeat** statement\_list **until** **(** logic\_expression **)**

24. for\_stmt → **for ID := int\_number to int\_number do** statement

25. input\_stmt → **readln** **(** var\_list **)**

26. output\_stmt → **writeln** **(** output\_list **)**

27. output\_list → output\_list **,** output | output

28. output → arithmetic\_expression | **STRING |** ε

29. var → **ID** | **ID** **[** arithmetic\_expression **]**

30. logic\_expression → arithmetic\_expression relop arithmetic\_expression

31. relop → **<=** | **<** | **>** | **>=** | **=** | **<>**

32. arithmetic\_expression → term | arithmetic\_expression **+** term | arithmetic\_expression **–** term

33. term → factor | term **\*** factor | term **/** factor

34. factor → **(** arithmetic\_expression **)** | var | call | **int\_number**  | **real\_number**

35. call → **ID (** args **)**

36. args → arg\_list | ε

37. arg\_list → arg\_list **,** arithmetic\_expression | arithmetic\_expression

## 2.2.- Recursividad por la izquierda

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

5. var\_declaration → var\_declaration var\_list : type\_specifier ; | var\_list : type\_specifier ;

A′= var\_declaration

α = var\_list : type\_specifier ;

β= var\_list : type\_specifier ;

var\_declaration → var\_list : type\_specifier ; var\_declaration′

var\_declaration′ → var\_list : type\_specifier ; var\_declaration′ | ε

6. var\_list → var\_list , ID | ID

A′= var\_list

α = , ID

β= ID

var\_list → ID var\_list′

var\_list′ → , ID var\_list′ |ε

10. functions\_block → functions\_block function\_declaration | ε

A′= functions\_block

α = function\_declaration

β= ε

functions\_block → functions\_block ′

functions\_block ′ → function\_declaration functions\_block ′ |ε

12. procedures\_block → procedures\_block procedure\_declaration | ε

A′= procedures\_block

α = procedure\_declaration

β= ε

procedures\_block→ ε procedures\_block′

procedures\_block′ → procedure\_declaration procedures\_block′ |ε

15. param\_list → param\_list var\_list : type\_specifier ; | var\_list : type\_specifier ;

A= param\_list

α = var\_list : type\_specifier ;

β= var\_list : type\_specifier ;

param\_list→ var\_list : type\_specifier ; param\_list′

param\_list′ → var\_list : type\_specifier ; param\_list′ | ε

18. statement\_list → statement\_list statement ; | statement ;

A′= statement\_list

α = statement ;

β= statement ;

statement\_list → statement ; statement\_list′

statement\_list′→ statement ; statement\_list′ | ε

27. output\_list → output\_list , output | output

A= output\_list

α = , output

β= output

output\_list → , output output\_list ′

output\_list ′→ , output output\_list ′ | ε

32. arithmetic\_expression → arithmetic\_expression + term | arithmetic\_expression - term | term

A= arithmetic\_expression

a1= + term

a2= - term

b= term

arithmetic\_expression → term arithmetic\_expression ′

arithmetic\_expression ′ → + term arithmetic\_expression ′ | - term arithmetic\_expression ′ | ε

33. term → term \* factor | term / factor | factor

A= term

a1= \* factor

a2= / factor

b= factor

term → factor term′

term′ → \* factor term′ | / factor term′ | ε

Eliminando la recursividad la gramática queda ahora de esta forma:

1. start → **program ID** **;** declaration\_block main\_block

2. declaration\_block → vars\_block functions\_block procedures\_block

3. main\_block → compound\_statement **•**

4. vars\_block → **var** var\_declaration | ε

5. var\_declaration → var\_list **:** type\_specifier **;** var\_declaration’

6. var\_declaration’ → var\_list **:** type\_specifier **;** var\_declaration’ | ε

7. var\_list → **ID** var\_list’

8. var\_list’ → **,** **ID** var\_list’ | ε

9. type\_specifier → basic\_type | array\_type

10. basic\_type → **integer** | **real** | **string**

11. array\_type → **array [ int\_number • • int\_number ]** **of** basic\_type

12. functions\_block → functions\_block’

13. functions\_block’ → function\_declaration functions\_block’ | ε

14. function\_declaration → **function ID(** params **)** **:** type\_specifier **;** local\_declarations compound\_stmt **;**

15. procedures\_block → procedures\_block’

16. procedures\_block’ → procedure\_declaration procedures\_block’ | ε

17. procedure\_declaration → **procedure ID(** params **) ;** local\_declarations compound\_stmt **;**

18. params → param\_list | ε

19. param\_list → var\_list **:** type\_specifier **;** param\_list’

20. param\_list’ → var\_list **:** type\_specifier **;** param\_list’ | ε

21. local\_declarations → vars\_block | ε

22. compound\_stmt → **begin** statement\_list **end**

23. statement\_list → statement **;** statement\_list’

24. statement\_list’ → statement **;** statement\_list’ | ε

25. statement → assignment\_stmt | call \_stmt | compound\_stmt | selection\_stmt | for\_stmt | repeat\_stmt | input\_stmt | output\_stmt

26. assignment\_stmt → var **:=** arithmetic\_expression | var **:= STRING**

27. call\_stmt → call

28. selection\_stmt → **if** **(** logic\_expression **)** then statement | **if** **(** logic\_expression **)** then statement else statement

29. repeat\_stmt → **repeat** statement\_list **until** **(** logic\_expression **)**

30. for\_stmt → **for ID := int\_number to int\_number do** statement **;**

31. input\_stmt → **readln** **(** var\_list **)**

32. output\_stmt → **writeln** **(** output\_list **)**

33. output\_list → output output\_list’

34. output\_list’ → **,** output output\_list’ | ε

35. output → arithmetic\_expression | **STRING** | ε

36. var → **ID** | **ID** **[** arithmetic\_expression **]**

37. logic\_expression → arithmetic\_expression relop arithmetic\_expression

38. relop → **<=** | **<** | **>** | **>=** | **=** | **<>**

39. arithmetic\_expression → term arithmetic\_expression’

40. arithmetic\_expression’ → + term arithmetic\_expression’| - term arithmetic\_expression’| ε

41. term → factor term’

42. term’ → \* factor term’ | / factor term’ | ε

43. factor → **(** arithmetic\_expression **)** | var | call | **int\_number** | **real\_number**

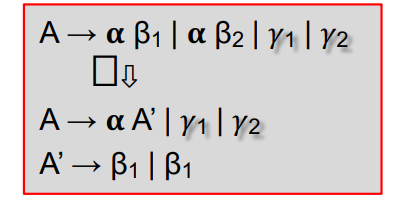
44. call → **ID (** args **)**

45. args → arg\_list | ε

46. arg\_list → arithmetic\_expression arg\_list’

47. arg\_list’ → **,** arithmetic\_expression arg\_list’ | ε

## 2.3.- Factorización a la izquierda



26. assignment\_stmt → var := arithmetic\_expression | var := STRING

Eliminate left factoring

A= assignment\_stmt ;

α =var :=

β1= arithmetic\_expression

β2= STRING

assignment\_stmt → var := assignment\_stmt ′

assignment\_stmt′ →arithmetic\_expression | STRING

29. selection\_stmt → if ( logic\_expression ) then statement

| if ( logic\_expression ) then statement else statement

A= selection\_stmt

α = if ( logic\_expression ) then statement

β1= ε

β2= else statement

selection\_stmt → if ( logic\_expression ) then statement selection\_stmt′

selection\_stmt′ → ε | else statement

36. var → ID | ID [ arithmetic\_expression ]

A= var

α = ID

β1= ε

β2= [ arithmetic\_expression ]

var → ID var′

var′ → ε | [ arithmetic\_expression ]

Simplificación de la gramática:

Quitamos funciones unitarias.

· start → **program ID** **;** declaration\_block main\_block

· declaration\_block → vars\_block functions\_block procedures\_block

· main\_block → compound\_statement **•**

· compound\_stmt → **begin** statement\_list **end**

o start → **program ID** **;** vars\_block functions\_block procedures\_block **begin** statement\_list **end •**

· vars\_block → **var** var\_declaration | ε

· var\_declaration → var\_list **:** type\_specifier **;** var\_declaration’

· var\_list → **ID** var\_list’

o vars\_block → **var** **ID** var\_list’ **:** type\_specifier **;** var\_declaration’ | ε

· type\_specifier → basic\_type | array\_type

· basic\_type → **integer** | **real** | **string**

· array\_type → **array [ int\_number• • int\_number]** **of** basic\_type

o type\_specifier → **integer** | **real** | **string** | **array [ int\_number• • int\_number]** **of**

basic\_type

· functions\_block’ → function\_declaration functions\_block’ | ε

· function\_declaration → **function ID(** params **)** **:** type\_specifier **;** local\_declarations

compound\_stmt **;**

· compound\_stmt → **begin** statement\_list **end**

· functions\_block’ → **function ID(** params **)** **:** type\_specifier **;** local\_declarations

**begin** statement\_list **end ;** functions\_block’ | ε

· procedures\_block’ → procedure\_declaration procedures\_block’ | ε

· procedure\_declaration → **procedure ID(** params **) ;** local\_declarations compound\_stmt **;**

· compound\_stmt → **begin** statement\_list **end**

o procedures\_block’ → **procedure ID(** params **) ;** local\_declarations **begin**

statement\_list **end ;** procedures\_block’ | ε

· params → param\_list | ε

· param\_list → var\_list **:** type\_specifier **;** param\_list’

· var\_list → **ID** var\_list’

o params → **ID** var\_list’ **:** type\_specifier **;** param\_list’ | ε

· statement → assignment\_stmt | call \_stmt | compound\_stmt | selection\_stmt | for\_stmt |

repeat\_stmt | input\_stmt | output\_stmt

· assignment\_stmt → var := assignment\_stmt’

· var → **ID** var′

· call\_stmt → call

· call → **ID (** args **)**

· compound\_stmt → **begin** statement\_list **end**

· selection\_stmt → **if** ( logic\_expression ) **then** statement selection\_stmt’

· for\_stmt → **for ID := int\_number** **to int\_number** **do** statement

· repeat\_stmt → **repeat** statement\_list **until** **(** logic\_expression **)**

· input\_stmt → **readln** **(** var\_list **)**

· output\_stmt → **writeln** **(** output\_list **)**

· output\_list → output output\_list’

o statement → **ID** var′ **:=** assignment\_stmt’ | **ID (** args **)** | **begin** statement\_list **end** | **if** ( logic\_expression ) **then** statement selection\_stmt’ | **for ID := NUMBER to NUMBER do** statement | **repeat** statement\_list **until** **(** logic\_expression **)**| **readln** **(** var\_list **)** | **writeln** **(** output output\_list’ **)**

· factor → **(** arithmetic\_expression **)** | var | call | **int\_number** | **real\_number**

· var → **ID** var′

· call\_stmt → call

· call → **ID (** args **)**

o factor → **(** arithmetic\_expression **)** | **ID** var′ | **ID (** args **)** | | **int\_number** | **real\_number**

# 3.- Diseño

## 3.1.- Gramática final

1. start → **program ID** **;** vars\_block functions\_block procedures\_block **begin** statement\_list **end •**

2. vars\_block → **var** **ID** var\_list’ **:** type\_specifier **;** var\_declaration’ | ε

3. var\_declaration’ → var\_list **:** type\_specifier **;** var\_declaration’ | ε

4. var\_list’ → **,** **ID** var\_list’ | ε

5. type\_specifier → **integer** | **real** | **string** | **array [ int\_number • • int\_number ]** **of** basic\_type

6. basic\_type → **integer** | **real** | **string**

7. functions\_block → functions\_block’

8. functions\_block’ → **function ID(** params **)** **:** type\_specifier **;** local\_declarations **begin** statement\_list **end ;** functions\_block’ | ε

9. procedures\_block → procedures\_block’

10. procedures\_block’ → **procedure ID(** params **) ;** local\_declarations **begin** statement\_list **end ;** procedures\_block’ | ε

11. params → **ID** var\_list’ **:** type\_specifier **;** param\_list’ | ε

12. param\_list’ → var\_list **:** type\_specifier **;** param\_list’ | ε

13. local\_declarations → vars\_block | ε

14. statement\_list → statement **;** statement\_list’

15. statement\_list’ → statement **;** statement\_list’ | ε

16. statement → **ID** statement′ | **begin** statement\_list **end**| **if (** logic\_expression **) then** statement selection\_stmt′ | **for ID := int\_number to int\_number do** statement | **repeat** statement\_list **until (** logic\_expression **)** | **readln ( ID** var\_list′ **);** | **writeln (** output output\_list′**) ;**

17. statement′ → var′ **:=**  assignment\_stmt′ | **(** args **)**

18. assignment\_stmt’ → arithmetic\_expression | STRING

19. selection\_stmt’ → **else** statement | ε

20. output\_list’ → **,** output output\_list’ | ε

21. output → arithmetic\_expression | **STRING** | ε

22. var′ → **[** arithmetic\_expression **]** | ε

23. logic\_expression → arithmetic\_expression relop arithmetic\_expression

24. relop → **<=** | **<** | **>** | **>=** | **=** | **<>**

25. arithmetic\_expression → term arithmetic\_expression’

26. arithmetic\_expression’ → + term arithmetic\_expression’| - term arithmetic\_expression’| ε

27. term → factor term’

28. term’ → \* factor term’ | **/** factor term’ | ε

29. factor → **ID** factor′ | **int\_number** | **real\_number** | **(**arithmetic\_operator**)**

30. factor′ → **(** args **)** | var′

31. args → arithmetic\_expression arg\_list’ | ε

32. arg\_list’ → **,** arithmetic\_expression arg\_list’ | ε

## 3.2.- Calculo de los First

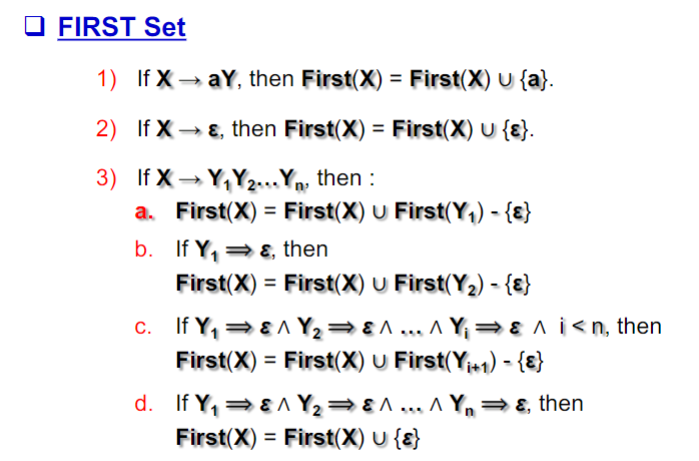


Figura 1

|  |  |
| --- | --- |
| **NT** | **FIRST** |
| start | **program** |
| vars\_block | **var** **ε** |
| var\_declaration′ | **ID ε** |
| var\_list′ | **, ε** |
| type\_specifier | **integer**  **real**  **string array** |
| basic\_type | **integer**  **real string** |
| functions\_block | **function** |
| functions\_block ′ | **function** **ε** |
| procedures\_block | **procedure** |
| procedures\_block′ | **procedure**  **ε** |
| params | **ID ε** |
| param\_list′ | **ID**  **ε** |
| local\_declarations | **var**  **ε** |
| statement\_list | **ID** **begin**  **if** **for**  **repeat** **readln**  **writeln** |
| statement\_list′ | **ID** **begin** **if** **for**  **repeat** **readln**  **writeln**  ε |
| statement | **ID**  **begin**  **if** **for**  **repeat** **readln**  **writeln** |
| statement′ | **[ ( ε** |
| assignment\_stmt′ | **ID** **real\_number** **int\_number**  **(**  **STRING** |
| selection\_stmt′ | **else**  **ε** |
| output\_list ′ | **,**  **ε** |
| output | **ID real\_number** **int\_number**  **(**  **STRING** , **ε** |
| var′ | **[ ε** |
| logic\_expression | **ID real\_number** **int\_number**  **(** |
| relop | **<=** , **<** , **>** , **>=** , **=** , **<>** |
| arithmetic\_expression | **ID real\_number** **int\_number**  **(** |
| arithmetic\_expression′ | **+**  **-**  **ε** |
| term | **ID real\_number** **int\_number**  **(** |
| term′ | **\***  **/**  **ε** |
| factor | **ID real\_number** **int\_number**  **(** |
| factor′ | **[**  **ε**  **(** |
| args | **ID real\_number** **int\_number**  **(**  **ε** |
| arg\_list ′ | **,** **ε** |

## 3.3.- Calculo de los Follow

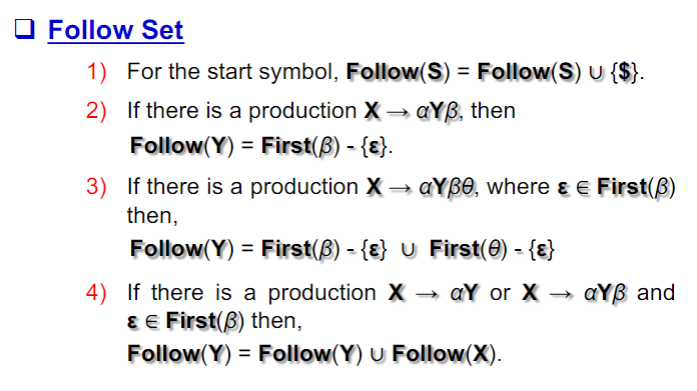


Figura 2

1. Follow(start) = {**$**}

2. Follow(vars\_block) = First(functions\_block) U First(procedures\_block ) U Follow(local\_declarations)

= {**function**} U {**procedure**} U {**begin**}

3. Follow(var\_declaration′) = Follow(vars\_block)

= {**function**} U {**procedure**} U {**begin**}

4. Follow(var\_list′) = Follow(vars\_block) U Follow(var\_declaration′) U Follow(params) U Follow(param\_list′)

= { **:** } U { **)** }

5. Follow(type\_specifier)

= { **;** }

6. Follow(basic\_type) = Follow(type\_specifier)

= { **;** }

7. Follow(functions\_block) = First(procedures\_block) - {ε} U {**begin**}

= {**procedure**} U {**begin**}

8. Follow(functions\_block′) = Follow(functions\_block)

= {**procedure**} U {**begin**}

9. Follow(procedures\_block)

= {**begin**}

10. Follow(procedures\_block′) = Follow (procedures\_block)

= {**begin**}

11. Follow(params)

= { **)** }

12. Follow(param\_list′) = Follow(params)

= { **)** }

13. Follow(local\_declarations)

= {**begin**}

14. Follow(statement\_list)

= {**end**} U {**until**}

15. Follow(statement\_list′) = Follow(statement\_list)

= {**end**} U {**until**}

16. Follow(statement) = { **;** } U Follow(selection\_stmt′)

= { **;** } U {**else**}

17. Follow(statement′) = Follow(statement)

= { **;** } U {**else**}

18. Follow(assignment\_stmt′) = Follow(statement′)

= { **;** } U {**else**}

19. Follow(selection\_stmt′)

= { **;** } U {**else**}

20. Follow(output\_list′)

= { **)** }

21. Follow(output) = First (output\_list′) - {ε}

= { **,** }

22. Follow(var′) = { **:=** } U Follow(factor′)

= { **:=** } U { **\*** } U { **/** }

23. Follow(logic\_expression)

= { **)** }

24. Follow(relop) = First(arithmetic\_expression) - {ε}

= { **ID** , **int\_number, real\_number**, **(** }

25. Follow(arithmetic\_expression) = Follow(assignment\_stmt′) U Follow(output) U { **]** } U First(relop) - {ε} U Follow(logic\_expression) U First(arg\_list′) - {ε} U Follow(arg) U Follow(arg\_list’) U Follow(logic\_expression) =

= { **]** } U { **;** } U {**else**} U { **,** } U { **<=** , **<** , **>** , **>=** , **=** , **<>** } U { **,** } U { **)** }

= { **]** , **;** , **else** , **,** ,**<=** , **<** , **>** , **>=** , **=** , **<>**, **)** }

26. Follow(arithmetic\_expression’) = Follow(arithmetic\_expression)

= { **]** , **;** , **else** , **,** ,**<=** , **<** , **>** , **>=** , **=** , **<>**, **)** }

27. Follow(term) = First(arithmetic\_expression′) - {ε}

= { **+** , **-** }

28. Follow(term′) = Follow(term)

= { **+** , **-** }

29. Follow(factor) = First(term′) - {ε}

= { **\*** }U { **/** }

30. Follow(factor′) = Follow(factor)

= { **\*** }U { **/** }

31. Follow(args)

= { **)** }

32. Follow(arg\_list′)= Follow(args)

= { **)** }

La tabla final quedaría de la siguiente forma:

|  |  |  |
| --- | --- | --- |
| **NT** | **FIRST** | **FOLLOW** |
| start | **program** | **$** |
| vars\_block | **var** **ε** | **function, procedure, begin** |
| var\_declaration′ | **ID ε** | **function, procedure, begin** |
| var\_list′ | **, ε** | **: , )** |
| type\_specifier | **integer**  **real**  **string array** | **;** |
| basic\_type | **integer**  **real string** | **;** |
| functions\_block | **function** | **procedure , begin** |
| functions\_block ′ | **function** **ε** | **procedure , begin** |
| procedures\_block | **procedure** | **begin** |
| procedures\_block′ | **procedure**  **ε** | **begin** |
| params | **ID ε** | **)** |
| param\_list′ | **ID**  **ε** | **)** |
| local\_declarations | **var**  **ε** | **begin** |
| statement\_list | **ID** **begin**  **if** **for**  **repeat** **readln**  **writeln** | **end , until** |
| statement\_list′ | **ID** **begin** **if** **for**  **repeat** **readln**  **writeln**  ε | **end , until** |
| statement | **ID**  **begin**  **if** **for**  **repeat** **readln**  **writeln** | **; , else** |
| statement′ | **[ ( ε** | **; , else** |
| assignment\_stmt′ | **ID** **real\_number** **int\_number**  **(**  **STRING** | **; , else** |
| selection\_stmt′ | **else**  **ε** | **; , else** |
| output\_list ′ | **,**  **ε** | **)** |
| output | **ID real\_number** **int\_number**  **(**  **STRING** , **ε** | **,** |
| var′ | **[ ε** | **:= \* / + -** |
| logic\_expression | **ID real\_number** **int\_number**  **(** | **)** |
| relop | **<=** , **<** , **>** , **>=** , **=** , **<>** | **ID real\_number** **int\_number (** |
| arithmetic\_expression | **ID real\_number** **int\_number**  **(** | **] ; else , <= < > >= = <> )** |
| arithmetic\_expression′ | **+**  **-**  **ε** | **] ; else , <= < > >= = <> )** |
| term | **ID real\_number** **int\_number**  **(** | **+ -** |
| term′ | **\***  **/**  **ε** | **+ -** |
| factor | **ID real\_number** **int\_number**  **(** | **\* , /** |
| factor′ | **[**  **ε**  **(** | **\* , /** |
| args | **ID real\_number** **int\_number**  **(**  **ε** | **)** |
| arg\_list ′ | **,** **ε** | **)** |

## 3.4.- Calculo de los First+

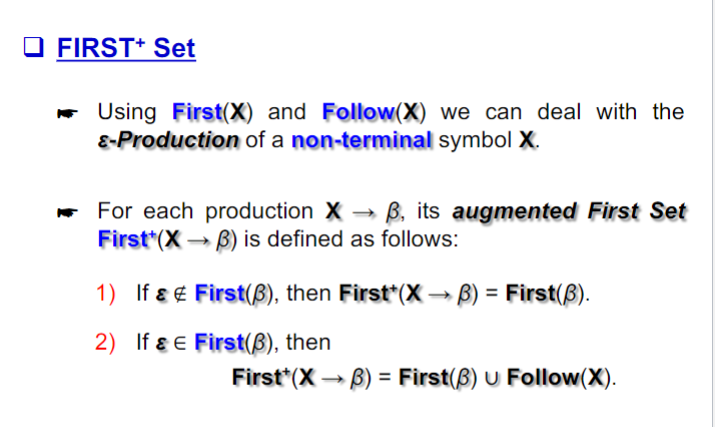


Figura 3

1. FirstPlus(start → **program ID** **;** vars\_block functions\_block procedures\_block **begin** statement\_list **end** •) = { **program** }

2. FirstPlus(vars\_block → **var ID** var\_list′ **:** type\_specifier **;** var\_declaration′) = {**var**}

FirstPlus (vars\_block → ε ) = { **function** , **procedure** , **begin** , **ε** }

3. FirstPlus(var\_declaration′ → **ID** var\_list′ **:** type\_specifier **;** var\_declaration′) = { **ID** }

FirstPlus (var\_declaration′ → ε) = { **function** , **procedure** , **begin** , **ε** }

4. FirstPlus(var\_list′ → **, ID** var\_list′) = { **,** }

FirstPlus (var\_list′ → ε) = { **:** , **)** ,ε }

5. FirstPlus(type\_specifier → **integer** | **real** | **string** | **array [ int\_number** • • **int\_number] of** basic\_type) = { **integer** , **real** , **string** , **array** }

6. FirstPlus(basic\_type → **integer** | **real** | **string**)= { **integer** , **real** , **string** }

7. FirstPlus(functions\_block → functions\_block′ ) = First(functions\_block′) ={ **function** }

8. FirstPlus(functions\_block′ → **function ID(** params **) :** type\_specifier **;** local\_declarations **begin** statement\_list **end;** functions\_block′) = { **function** }

FirstPlus (functions\_block′ → ε) U { **ε** } = { **procedure** , **begin** , **ε** }

9. FirstPlus(procedures\_block→ procedures\_block′ ) = First(procedures\_block′) = { **procedure** }

10. FirstPlus(procedures\_block′ → **procedure ID(** params **) ;** local\_declarations **begin** statement\_list **end ;** procedures\_block′) = { **procedure** }

FirstPlus (procedures\_block′ → ε) U { **ε** } = { **begin** , **ε** }

11. FirstPlus(params → **ID** var\_list′ **:** type\_specifier **;**  param\_list′) = { **ID** }

FirstPlus (params → ε) = { **)** , **ε** }

12. FirstPlus(param\_list′ → **ID** var\_list′ **:** type\_specifier **;**  param\_list′) = { **ID** }

FirstPlus (param\_list′ → ε) = { **)** , **ε** }

13. FirstPlus(local\_declarations → vars\_block ) = first( vars\_block ) = { **var** , ε }

FirstPlus (local\_declarations→ ε ) ={ **begin** , **ε** }

14. FirstPlus(statement\_list → statement **;** statement\_list′) = First(statement\_list) = { **ID** , **begin** , **if** , **for** , **repeat** , **readln** , **writeln** }

15. FirstPlus(statement\_list′→ statement **;** statement\_list′ | ε) = { **ID**, **begin**, **if** , **for** , **repeat** , **readln** , **writeln**} U Follow(statement\_list′) = { **ID** , **begin** , **if** , **for** , **repeat** , **readln** , **writeln** , **end** ,**until** }

16. FirstPlus(statement → **ID** statement′ | **begin** statement\_list **end**| **if (** logic\_expression **) then** statement selection\_stmt′ | **for ID := int\_number** **to int\_number do** statement | **repeat** statement\_list **until (** logic\_expression **)**| **readln ( ID** var\_list′ **);** | **writeln ( ,** output output\_list′**);**) = First(statement) = { **ID**, **begin** , **if** , **for** , **repeat** , **readln** , **writeln** }

17. FirstPlus(statement′ → var′ **:=**  assignment\_stmt′ | **(** args **)**) = First(var’) = { **[** ,ε , **(** }

18. FirstPlus(assignment\_stmt′ → arithmetic\_expression | **STRING** ) = First(assignment\_stmt′) = { **ID** , **real\_number , int\_number**, **(** , **STRING** }

19. FirstPlus(selection\_stmt′ → **else** statement ) = { **else** }

FirstPlus(selection\_stmt′→ ε) = { **;** }

20. FirstPlus(output\_list′→ **,** output output\_list′) = { **,** }

FirstPlus(output\_list′→ε ) ={ **)** }

21. FirstPlus(output → arithmetic\_expression | **STRING |** ε) = First(arithmetic\_expression)= { **STRING** } U { **ID**, **real\_number , int\_number**, **(** }

FirstPlus(output →ε) = { **,** }

22. FirstPlus(var′ → **[** arithmetic\_expression **]**) = { **[** }

Follow(var′→ ε) = { **:=** , **\*** , **/** , **ε** }

23. FirstPlus(logic\_expression → arithmetic\_expression relop arithmetic\_expression) = First(arithmetic\_expression) = { **ID**, **real\_number , int\_number**, **(** }

24. FirstPlus(relop → **<=** | **<** | **>** | **>=** | **=** | **<>** ) = {**<=** , **<** , **>** , **>=** , **=** , **<>**}

25. FirstPlus(arithmetic\_expression → term arithmetic\_expression′) = First(term) = { **ID** , **real\_number , int\_number**, **(** }

26. FirstPlus(arithmetic\_expression′ → **+** term arithmetic\_expression′ | **-** term arithmetic\_expression′) = { **+** , **-** }

FirstPlus(arithmetic\_expression′→ ε) = { **]** , **;** , **else** , **,** ,**<=** , **<** , **>** , **>=** , **=** , **<>**, **) ,** ε }

27. FirstPlus(term → factor term′) = First(factor) = { **ID** , **real\_number , int\_number**,  **(** }

28. FirstPlus(term′ → **\*** factor term′ | **/** factor term′) = { \* , **/** }

FirstPlus(term′→ ε) = { **+** , **-** }

29. FirstPlus(factor → **ID** factor′ | **real\_number | int\_number**| **(**arithmetic\_operator**)**) = { **ID** , **real\_number , int\_number**, **(** }

30. FirstPlus( factor′ → **(** args **)** | var′ ) = { **(** } U First(var’) ={ **(** , **[** , **ε** }

31. FirstPlus(args → arithmetic\_expression arg\_list′) = First(arithmetic\_expression) = { **ID** , **real\_number , int\_number**, **(** }

FirstPlus(args→ ε) = { **)** , ε }

32. FirstPlus(arg\_list′→ **,** arithmetic\_expression arg\_list ′ | ε )={ **,** }

FirstPlus(arg\_list′→ ε) = { **)** , **ε** }

# 4.- Implementación

Creamos la función agregar\_token() la cual recibe el id del token para guardar la secuencia de tokens. (Figura 1)

Texto

Descripción generada automáticamente

Figura 4

Nuestra función Main quedaría de la siguiente forma:

Como se puede apreciar en la Figura 5 la función parser() agrega el símbolo “$” al final de la secuencia de tokens e inicializamos una posición en 0 para ir recorriendo los tokens. Después mandamos a llamar el método del símbolo inicial de la gramática en este caso start() le pasamos como parámetro current\_token una vez termina la tarea de start() comprobamos si el elemento de la lista de tonkes es igual a símbolo de $ si se cumple quieres decir que no hubieron errores sintácticos. Si no se cumple quiere decir que hubo un error en el vaciado de la lista.

Texto

Descripción generada automáticamente

Figura 5

Ya estando en el método start vamos a evaluar su secuencia acorde a la gramática definitiva si el primer valor de la lista es igual a program avanzamos con el analizador sumando a la posición más 1. Si eso no se cumple mandamos el primer mensaje de error. Así seguiremos avanzando poco a poco hasta llegar a un símbolo no terminal si es un símbolo no terminal llamamos a la función correspondiente.

Texto

Descripción generada automáticamente

Figura 6

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 7

Si en dado caso llegamos a un símbolo terminal tenemos que evaluar con los if y si la tenemos que enviarlo a una función match que se va a encargar de hacer la comprobación de que sea el símbolo correcto en el lugar adecuado si es así mandamos a llamar otra posición. Si alguna de las producciones de la gramática tenía épsilon lo quitamos con la unión de vacío con el conjunto del firstplus de la producción.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Figura 8

# 5.- Verificación y Validación

No se completó con éxito el Código por lo que no hubo parte de testing.

# 6.- Referencias

1.- R. Castelló, Class Lecture, Topic: “Chapter 2 – Lexical Analysis.” TC3048, School of Engineering and Science,ITESM, Chihuahua, Chih, April, 2020. 14

2.- Museu informática, “El Lenguaje de Programación Pascal”, S.F; disponible en:<https://museo.inf.upv.es/pascal/> ; Internet; consultado el 12 de octubre 2023

3.- ¿Qué es un analizador sintáctico? - Ryte Marketing Wiki. (2023.).<https://es.ryte.com/wiki/Analizador_Sint%C3%A1ctico> Internet; consultado el 12 de octubre 2023