



Desarrollo y Análisis de un Parser en Lenguaje C--

Instituto Tecnológico y de
Estudios Superiores de
Monterrey, Campus Chihuahua

Profesor

Dr. Rodolfo Castelló Zetina

Clase

Diseño de Compiladores

Alumno

Sebastian Salazar Villanueva

A01568158

Fecha

24 de Junio del 2022

Índice

1. Introducción.	3
1.1. Resumen.	3
1.2. Notación.	3
2. Análisis.	3
2.1. Requerimientos	3
2.2. Gramática.	5
3. Diseño.	39
3.1. Pseudocódigo	39
3.2. Tabla de Parsing.	39
3.3. Tablas de Símbolos	40
4. Implementación	40
4.1. Código Fuente	40
5. Verificación y Validación.	66
5.1. Casos de Prueba	66
6. Referencias.	68

1. Introducción

1.1. Resumen

En esta sección se proporcionará seguimiento de desarrollo del escáner en el lenguaje C++, esta vez correspondiendo a el parser y dándole seguimiento a este mismo. Se proporcionará un documento de Especificación de Requisitos de Software(ERS), dividido en 5 varias secciones:

Introducción, Análisis, Diseño, Implementación, Verificación y Validación, Referencias

1.2. Notación

Tabla de Parsing: Tabla en la que se manejan los no terminales en la primera columna y después de manejan los terminales en la primera fila, añadiendo el símbolo de '\$', de esta forma se empieza a trabajar sobre cada casilla con los First+ correspondientes de cada producción a la izquierda con sus respectivos terminales.

Producciones: Una producción es aquella definida por algún caracter o caracteres que produce caracteres terminales o no terminales, siendo estos últimos llamadas a otras producciones.

2. Análisis

2.1. Requerimientos

- El sistema deberá de realizarse en el lenguaje de programación C++.
- El sistema deberá arrojar un mensaje de 'OK' cuando no exista ningún error.
- El sistema se desarrollará con el algoritmo de Descenso Recursivo.
- El sistema deberá de arrojar las tablas de símbolos actualizadas.
- En caso de errores, estos deberán especificarse para así tener una mejor comprensión para corregir el error que despliega el sistema.
- El sistema deberá arrojar el número de funciones totales declaradas.
- El sistema deberá arrojar el número de variables locales totales declaradas.
- El sistema deberá de arrojar el número de variables globales totales declaradas.
- El sistema deberá de arrojar un número de Llamadas a función totales.
- El sistema deberá de arrojar un número de uso de variables locales.
- El sistema deberá de arrojar el nombre de sus funciones junto a su tipo de función y número de parámetros.

- El sistema deberá arrojar un error si existe más de una variable global con el mismo nombre.
- El sistema deberá arrojar un error si existe más de una función con el mismo nombre.
- El sistema deberá arrojar un error si la función con el nombre “main” o “MAIN” no es lo último declarado.
- Lista de Tokens Válidos:
 1. **RETURN**
 2. **WHILE**
 3. **VOID**
 4. **ELSE**
 5. **IF**
 6. **INT**
 7. **INPUT**
 8. **OUTPUT**
 9. **>**
 10. **<**
 11. **=**
 12. **<=**
 13. **==**
 14. **>=**
 15. **!=**
 16. **(**
 17. **)**
 18. **[**
 19. **]**
 20. **{**
 21. **}**
 22. **/**
 23. *****
 24. **-**
 25. **+**
 26. **,**
 27. **;**
 28. **Identificador**
 29. **Constante Entera**
 30. **\$**

2.2. Gramática

2- Producciones

1. program --> declaration_list
2. declaration_list --> declaration_list declaration | declaration
3. declaration --> var_declaration | fun_declaration
4. var_declaration --> type_specifier **ID** ; | type_specifier **ID** [**NUM**] ;
5. type_specifier --> **int** | **void**
6. fun_declaration --> type_specifier **ID** (params) compound_stmt
7. params --> param_list | **void**
8. param_list --> param_list , param | param
9. param --> type_specifier **ID** | type_specifier **ID** []
10. compound_stmt --> { local_declarations statement_list }
11. local_declarations --> local_declarations var_declaration | e
12. statement_list --> statement_list statement | e
13. statement --> assignment_stmt | call_stmt | compound_stmt | selection_stmt
| iteration_stmt | return_stmt | input_stmt | output_stmt
14. assignment_stmt --> var = expression ;
15. call_stmt --> call ;
16. selection_stmt --> **if** (expression) statement | **if** (expression) statement **else** statement
17. iteration_stmt --> **while** (expression) statement
18. return_stmt --> **return** ; | **return** expression ;
19. input_stmt --> **input** var ;
20. output_stmt --> **output** expression ;
21. var --> **ID** | **ID** [arithmetic_expression]

- 22. expression --> arithmetic_expression relop arithmetic_expression
| arithmetic_expression
- 23. relop --> <= | < | > | >= | == | !=
- 24. arithmetic_expression --> arithmetic_expression addop term | term
- 25. addop --> + | -
- 26. term --> term mulop factor | factor
- 27. mulop --> * | /
- 28. factor --> (arithmetic_expression) | var | call | **NUM**
- 29. call --> **ID** (args)
- 30. args --> args_list | **e**
- 31. args_list --> args_list , arithmetic_expression | arithmetic_expression

Errores(1)

- 1. program --> declaration_list void ID(void) | void ID(void)
- 2. declaration_list --> declaration_list declaration | declaration
- 3. declaration --> var_declaration | fun_declaration

Errores(2)

- 4. var_declaration --> **int ID ;** | **int ID [NUM] ;**
- 5. type_specifier --> **int** | **void**

Errores(3)

- 6. fun_declaration --> type_specifier **ID** (params) compound_stmt
- 7. params --> param_list | **void**
- 8. param_list --> param_list , param | param
- 9. param --> **int ID** | **int ID []**

2-Producciones sin errores

1. program --> declaration_list **void ID(void)** compound_stmt | **void ID(void)** compound_stmt
2. declaration_list --> declaration_list declaration | declaration
3. declaration --> var_declaration | fun_declaration
4. var_declaration --> **int ID ;** | **int ID [NUM] ;**
5. type_specifier --> **int** | **void**
6. fun_declaration --> type_specifier **ID** (params) compound_stmt
7. params --> param_list | **void**
8. param_list --> param_list , param | param
9. param --> **int ID** | **int ID []**
10. compound_stmt --> { local_declarations statement_list }
11. local_declarations --> local_declarations var_declaration | e
12. statement_list --> statement_list statement | e
13. statement --> assignment_stmt | call_stmt | compound_stmt | selection_stmt
| iteration_stmt | return_stmt | input_stmt | output_stmt
14. assignment_stmt --> var = expression ;
15. call_stmt --> call ;
16. selection_stmt --> **if** (expression) statement | **if** (expression) statement **else** statement
17. iteration_stmt --> **while** (expression) statement
18. return_stmt --> **return ;** | **return** expression ;
19. input_stmt --> **input** var ;
20. output_stmt --> **output** expression ;
21. var --> **ID** | **ID** [arithmetic_expression]
22. expression --> arithmetic_expression relop arithmetic_expression | arithmetic_expression

23. relop --> <= | < | > | >= | == | !=

24. arithmetic_expression --> arithmetic_expression addop term | term

25. addop --> + | -

26. term --> term mulop factor | factor

27. mulop --> * | /

28. factor --> (arithmetic_expression) | var | call | **NUM**

29. call --> **ID** (args)

30. args --> args_list | **e**

31. args_list --> args_list , arithmetic_expression | arithmetic_expression

2- Recursividad Izquierda

Color Verde = Producciones que no se modificaron

Color Amarillo = Producciones que se modificaron

1. program --> declaration_list **void ID(void)** compound_stmt | **void ID(void)** compound_stmt

2. declaration_list --> declaration declaration_list '

3. declaration_list ' --> declaration declaration_list ' | **ε**

4. declaration --> var_declaration | fun_declaration

5. var_declaration --> **int ID ;** | **int ID [NUM] ;**

6. type_specifier --> **int** | **void**

7. fun_declaration --> type_specifier **ID** (params) compound_stmt

8. params --> param_list | **void**

9. param_list --> param param_list '

10. param_list ' --> , param param_list ' | **ε**

11. param --> **int ID** | **int ID []**

12. compound_stmt --> { local_declarations statement_list }

13. local_declarations --> local_declarations '
14. local_declarations ' --> var_declaration local_declarations ' | ϵ
15. statement_list --> statement_list '
16. statement_list ' --> statement statement_list ' | ϵ
17. statement --> assignment_stmt | call_stmt | compound_stmt | selection_stmt
| iteration_stmt | return_stmt | input_stmt | output_stmt
18. assignment_stmt --> var = expression ;
19. call_stmt --> call ;
20. selection_stmt --> if (expression) statement | if (expression) statement else statement
21. iteration_stmt --> while (expression) statement
22. return_stmt --> return ; | return expression ;
23. input_stmt --> input var ;
24. output_stmt --> output expression ;
25. var --> ID | ID [arithmetic_expression]
26. expression --> arithmetic_expression relop arithmetic_expression | arithmetic_expression
27. relop --> <= | < | > | >= | == | !=
28. arithmetic_expression --> term arithmetic_expression '
29. arithmetic_expression ' --> addop term arithmetic_expression ' | ϵ
30. addop --> + | -
31. term --> factor term '
32. term ' --> mulop factor term ' | ϵ
33. mulop --> * | /
34. factor --> (arithmetic_expression) | var | call | NUM
35. call --> ID (args)

36. args --> args_list | ϵ

37. args_list --> arithmetic_expression args_list '

38. args_list ' --> , arithmetic_expression args_list ' | ϵ

3- Factorización Izquierda

Color Verde = Producciones que no se modificaron

Color Amarillo = Producciones que se modificaron

1. program --> declaration_list **void ID(void)** compound_stmt | **void ID(void)** compound_stmt

2. declaration_list --> declaration declaration_list '

3. declaration_list ' --> declaration declaration_list ' | ϵ

4. declaration --> var_declaration | fun_declaration

5. var_declaration --> **int ID** var_declaration '

6. var_declaration ' --> ; | **[NUM] ;**

7. type_specifier --> **int** | **void**

8. fun_declaration --> type_specifier **ID** (params) compound_stmt

9. params --> param_list | **void**

10. param_list --> param param_list '

11. param_list ' --> , param param_list ' | ϵ

12. param --> **int ID** param '

13. param ' --> ϵ | **[]**

14. compound_stmt --> { local_declarations statement_list }

15. local_declarations --> local_declarations '

16. local_declarations ' --> var_declaration local_declarations ' | ϵ

17. statement_list --> statement_list '

18. statement_list ' --> statement statement_list ' | ϵ

19. statement --> assignment_stmt | call_stmt | compound_stmt | selection_stmt

| iteration_stmt | return_stmt | input_stmt | output_stmt

20. assignment_stmt --> var = expression ;

21. call_stmt --> call ;

22. selection_stmt --> if (expression) statement selection_stmt '

23. selection_stmt ' --> else statement | ϵ

24. iteration_stmt --> while (expression) statement

25. return_stmt --> return return_stmt '

26. return_stmt ' --> ; | expression ;

27. input_stmt --> input var ;

28. output_stmt --> output expression ;

29. var --> ID var '

30. var ' --> [arithmetic_expression] | ϵ

31. expression --> arithmetic_expression expression '

32. expression ' --> relop arithmetic_expression | ϵ

33. relop --> <= | < | > | >= | == | !=

34. arithmetic_expression --> term arithmetic_expression '

35. arithmetic_expression ' --> addop term arithmetic_expression ' | ϵ

36. addop --> + | -

37. term --> factor term '

38. term ' --> mulop factor term ' | ϵ

39. mulop --> * | /

40. factor --> (arithmetic_expression) | var | call | NUM

41. call --> ID (args)

42. args --> args_list | ϵ

43. args_list --> arithmetic_expression args_list '

44. args_list ' --> , arithmetic_expression args_list ' | ϵ

4- Eliminacion de producciones- ϵ

Color Verde = Producciones que no se modificaron

Color Amarillo = Producciones que se modificaron

Color Azul = Factor Izquierdo

Estados que derivan a ϵ { declaration_list ' , param_list ' , local_declarations ' , statement_list ' , selection_stmt ' , var ' , expression ' , arithmetic_expression ' , term ' , args , args_list ' }

1. program --> declaration_list **void ID(void)** compound_stmt | **void ID(void)** compound_stmt

2. declaration_list --> declaration declaration_list '

3. declaration_list ' --> declaration declaration_list ' | ϵ

4. declaration --> var_declaration | fun_declaration

5. var_declaration --> **int ID** var_declaration '

6. var_declaration ' --> ; | [**NUM**] ;

7. type_specifier --> **int** | **void**

8. fun_declaration --> type_specifier **ID** (params) compound_stmt

9. params --> param_list | **void**

10. param_list --> param param_list '

11. param_list ' --> , param param_list ' | ϵ

12. param --> **int ID** param '

13. param ' --> ϵ | []

14. compound_stmt --> { local_declarations statement_list }

15. local_declarations --> local_declarations '

16. local_declarations ' --> var_declaration local_declarations ' | ϵ
17. statement_list --> statement_list '
18. statement_list ' --> statement statement_list ' | ϵ
19. statement --> assignment_stmt | call_stmt | compound_stmt | selection_stmt
| iteration_stmt | return_stmt | input_stmt | output_stmt
20. assignment_stmt --> var = expression ;
21. call_stmt --> call ;
22. selection_stmt --> if (expression) statement selection_stmt '
23. selection_stmt ' --> else statement | ϵ
24. iteration_stmt --> while (expression) statement
25. return_stmt --> return return_stmt '
26. return_stmt ' --> ; | expression ;
27. input_stmt --> input var ;
28. output_stmt --> output expression ;
29. var --> ID var '
30. var ' --> [arithmetic_expression] | ϵ
31. expression --> arithmetic_expression expression '
32. expression ' --> relop arithmetic_expression | ϵ
33. relop --> <= | < | > | >= | == | !=
34. arithmetic_expression --> term arithmetic_expression '
35. arithmetic_expression ' --> addop term arithmetic_expression ' | ϵ
36. addop --> + | -
37. term --> factor term '
38. term ' --> mulop factor term ' | ϵ

39. mulop --> * | /

40. factor --> (arithmetic_expression) | var | call | NUM

41. call --> ID (call '

42. call ' --> args) |)

43. args --> args_list

44. args_list --> arithmetic_expression args_list '

45. args_list ' --> , arithmetic_expression args_list ' | ε

5- Eliminacion de producciones unitarias

Color Verde = Producciones que no se modificaron

Color Amarillo = Producciones que se modificaron

Color Azul = Factor Izquierdo

1. program --> declaration_list void ID(void) compound_stmt | void ID(void) compound_stmt

2. declaration_list --> declaration declaration_list '

3. declaration_list ' --> declaration declaration_list ' | ε

4. declaration --> int ID var_declaration ' | type_specifier ID (params) compound_stmt

5. var_declaration ' --> ; | [NUM] ;

6. type_specifier --> int | void

7. params --> param param_list ' | void

8. param_list ' --> , param param_list ' | ε

9. param --> int ID param '

10. param ' --> ε | []

11. compound_stmt --> { local_declarations statement_list }

12. local_declarations --> int ID var_declaration ' local_declarations | ε

13. statement_list --> statement statement_list | ε

14. statement --> **ID** statement ' { local_declarations statement_list } | **if** (expression) statement selection_stmt ' | **while** (expression) statement | **return** return_stmt ' | **input ID** var ' ; | **output** expression ;

Factor izq

15. statement ' --> var ' = expression ; | (call ' ;

16. selection_stmt ' --> **else** statement | ϵ

17. return_stmt ' --> ; | expression ;

18. var ' --> [arithmetic_expression] | ϵ

19. expression --> arithmetic_expression expression '

20. expression ' --> relop arithmetic_expression | ϵ

21. relop --> <= | < | > | >= | == | !=

22. arithmetic_expression --> term arithmetic_expression '

23. arithmetic_expression ' --> addop term arithmetic_expression ' | ϵ

24. addop --> + | -

25. term --> factor term '

26. term ' --> mulop factor term ' | ϵ

27. mulop --> * | /

29. factor --> (arithmetic_expression) | **ID** factor ' | **NUM**

Factor izq

29. factor ' --> var ' | (call ' ;

30. call ' --> args) |)

31. args --> arithmetic_expression args_list '

32. args_list ' --> , arithmetic_expression args_list ' | ϵ

6- Sustitución de producciones

Color Verde = Producciones que no se modificaron

Color Amarillo = Producciones que se modificaron

Color Morado = Caso de Factor Infinito

1. `program --> int ID declaration ' declaration_list ' | void ID (program '`

2. `program ' --> param param_list ') { local_declarations statement_list } declaration_list ' | void) { local_declarations statement_list } program "`

3. `program "" --> declaration_list ' | ε`

`program ' --> int ID param ' param_list ') { local_declarations statement_list }
declaration_list ' void ID(void) { local_declarations statement_list } | void) {
local_declarations statement_list } program "`

`program "" --> int ID declaration ' declaration_list ' void ID(void) { local_declarations
statement_list } | void ID (program ""`

Factor izq

`program "" --> param param_list ') { local_declarations statement_list } declaration_list ' void
ID(void) { local_declarations statement_list } | void) { local_declarations statement_list }
program ""`

Factor izq

`program "" --> declaration_list ' void ID(void) { local_declarations statement_list } | ε`

4. `declaration_list ' --> int ID declaration ' declaration_list ' | void ID (params) {
local_declarations statement_list } declaration_list ' | ε`

5. `declaration ' --> ; | [NUM] ; | (params) { local_declarations statement_list }`

6. `var_declaration ' --> ; | [NUM] ;`

7. `params --> param param_list ' | void`

8. `param_list ' --> , param param_list ' | ε`

9. param --> **int ID** param '

10. param ' --> ϵ | []

11. local_declarations --> **int ID** var_declaration ' local_declarations | ϵ

12. statement_list --> **ID** statement ' statement_list | { local_declarations statement_list } statement_list | **if** (expression) statement selection_stmt ' statement_list | **while** (expression) statement statement_list | **return** return_stmt ' statement_list | **input ID** var ' ; statement_list | **output** expression ; statement_list | ϵ

13. statement ' --> [arithmetic_expression] = expression ; | = expression ; | (call ' ;

14. statement --> **ID** statement ' | { local_declarations statement_list } | **if** (expression) statement selection_stmt ' | **while** (expression) statement | **return** return_stmt ' | **input ID** var ' ; | **output** expression ;

15. var ' --> [arithmetic_expression] | ϵ

16. selection_stmt ' --> **else** statement | ϵ

17. return_stmt ' --> ; | arithmetic_expression expression ' ;

18. expression --> arithmetic_expression expression '

19. expression ' --> <= arithmetic_expression | < arithmetic_expression | > arithmetic_expression | >= arithmetic_expression | == arithmetic_expression | != arithmetic_expression | ϵ

20. arithmetic_expression --> (arithmetic_expression) term ' arithmetic_expression ' | **ID** factor ' term ' arithmetic_expression ' | **NUM** term ' arithmetic_expression '

21. arithmetic_expression ' --> + term arithmetic_expression ' | - term arithmetic_expression ' | ϵ

22. term --> (arithmetic_expression) term ' | **ID** factor ' term ' | **NUM** term '

23. term ' --> mulop factor term ' | ϵ

24. mulop --> * | /

25. factor --> (arithmetic_expression) | **ID** factor ' | **NUM**

26. factor ' --> [arithmetic_expression] | ϵ | (call '

27. call ' --> args) |)

28. args --> arithmetic_expression args_list '

29. $\text{args_list}' \rightarrow , \text{arithmetic_expression } \text{args_list}' \mid \epsilon$

7- First

Color Amarillo = First de Producciones

1. $\text{program} \rightarrow \text{int ID declaration}' \text{ declaration_list}' \mid \text{void ID (program}'$

$\text{First}(\text{program}) = \{ \text{int}, \text{void} \}$

2. $\text{program}' \rightarrow \text{param param_list}' \{ \text{local_declarations statement_list} \} \text{ declaration_list}' \mid \text{void} \{ \text{local_declarations statement_list} \} \text{ program}''$

$\text{First}(\text{program}') = \{ \text{First}(\text{param}) - \{ \epsilon \} \} \cup \{ \text{void} \} = \{ \text{int}, \text{void} \}$

3. $\text{program}'' \rightarrow \text{declaration_list}' \mid \epsilon$

$\text{First}(\text{program}'') = \{ \text{First}(\text{declaration_list}') - \{ \epsilon \} \} \cup \{ \epsilon \} = \{ \text{int}, \text{void}, \epsilon \}$

4. $\text{declaration_list}' \rightarrow \text{int ID declaration}' \text{ declaration_list}' \mid \text{void ID (params)} \{ \text{local_declarations statement_list} \} \text{ declaration_list}' \mid \epsilon$

$\text{First}(\text{declaration_list}') = \{ \text{int}, \text{void}, \epsilon \}$

5. $\text{declaration}' \rightarrow ; \mid [\text{NUM}] ; \mid (\text{params}) \{ \text{local_declarations statement_list} \}$

$\text{First}(\text{declaration}') = \{ ; , [, (\}$

6. $\text{var_declaration}' \rightarrow ; \mid [\text{NUM}] ;$

$\text{First}(\text{var_declaration}') = \{ ; , [\}$

7. $\text{params} \rightarrow \text{param param_list}' \mid \text{void}$

$\text{First}(\text{params}) = \{ \text{First}(\text{param}) - \{ \epsilon \} \} \cup \{ \text{void} \} = \{ \text{int}, \text{void} \}$

8. $\text{param_list}' \rightarrow , \text{param param_list}' \mid \epsilon$

$\text{First}(\text{param_list}') = \{ , , \epsilon \}$

9. $\text{param} \rightarrow \text{int ID param}'$

$\text{First}(\text{param}) = \{ \text{int} \}$

10. $\text{param}' \rightarrow \epsilon \mid []$

$\text{First}(\text{param } ') = \{ [, \epsilon \}$

11. $\text{local_declarations} \rightarrow \text{int ID var_declaration } ' \text{local_declarations } | \epsilon$

$\text{First}(\text{local_declarations}) = \{ \text{int}, \epsilon \}$

12. $\text{statement_list} \rightarrow \text{ID statement } ' \text{statement_list } | \{ \text{local_declarations statement_list } \}$
 $\text{statement_list } | \text{if (expression) statement selection_stmt } ' \text{statement_list } | \text{while (}$
 $\text{expression) statement statement_list } | \text{return return_stmt } ' \text{statement_list } | \text{input ID var } ' ;$
 $\text{statement_list } | \text{output expression ; statement_list } | \epsilon$

$\text{First}(\text{statement_list}) = \{ \text{ID}, \{ , \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \epsilon \}$

13. $\text{statement } ' \rightarrow [\text{arithmetic_expression}] = \text{expression} ; | = \text{expression} ; | (\text{call } ' ;$

$\text{First}(\text{statement } ') = \{ [, = , (\}$

14. $\text{statement} \rightarrow \text{ID statement } ' | \{ \text{local_declarations statement_list } \} | \text{if (expression)}$
 $\text{statement selection_stmt } ' | \text{while (expression) statement } | \text{return return_stmt } ' | \text{input ID}$
 $\text{var } ' ; | \text{output expression ;}$

$\text{First}(\text{statement }) = \{ \text{ID}, \{ , \text{if}, \text{while}, \text{return}, \text{input}, \text{output} \}$

15. $\text{var } ' \rightarrow [\text{arithmetic_expression}] | \epsilon$

$\text{First}(\text{var } ') = \{ [, \epsilon \}$

16. $\text{selection_stmt } ' \rightarrow \text{else statement } | \epsilon$

$\text{First}(\text{selection_stmt } ') = \{ \text{else}, \epsilon \}$

17. $\text{return_stmt } ' \rightarrow ; | \text{arithmetic_expression expression } ' ;$

$\text{First}(\text{return_stmt } ') = \{ ; \} \cup \{ \text{First}(\text{arithmetic_expression}) - \{ \epsilon \} \} \cup \{ \text{First}(\text{expression } ') - \{ \epsilon \} \} = \{ (,$
 $\text{ID}, \text{NUM}, ; , <= , < , > , >= , == , != , \epsilon \}$

18. $\text{expression} \rightarrow \text{arithmetic_expression expression } ' \}$

$\text{First}(\text{expression}) = \{ \text{First}(\text{arithmetic_expression}) - \{ \epsilon \} \} \cup \{ \text{First}(\text{expression } ') - \{ \epsilon \} \} = \{ (, \text{ID},$
 $\text{NUM}, <= , < , > , >= , == , != , \epsilon \}$

19. $\text{expression } ' \rightarrow <= \text{arithmetic_expression } | < \text{arithmetic_expression } | >$
 $\text{arithmetic_expression } | >= \text{arithmetic_expression } | == \text{arithmetic_expression } | !=$
 $\text{arithmetic_expression } | \epsilon$

$\text{First}(\text{expression } ') = \{ <= , < , > , >= , == , != , \epsilon \}$

20. arithmetic_expression \rightarrow (arithmetic_expression) term ' arithmetic_expression ' | ID factor ' term ' arithmetic_expression ' | **NUM** term ' arithmetic_expression '

First(arithmetic_expression) = { (, ID , NUM }

21. arithmetic_expression ' \rightarrow + term arithmetic_expression ' | - term arithmetic_expression ' | ϵ

First(arithmetic_expression ') = { + , - , ϵ }

22. term \rightarrow (arithmetic_expression) term ' | ID factor ' term ' | **NUM** term '

First(term) = { (, ID , NUM }

23. term ' \rightarrow mulop factor term ' | ϵ

First(term ') = {First(mulop) - { ϵ }} \cup {First(factor) - { ϵ }} \cup {First(term ') - { ϵ }} \cup { ϵ } = { * , / , ϵ , (, ID , NUM }

24. mulop \rightarrow * | /

First(mulop) = { * , / }

25. factor \rightarrow (arithmetic_expression) | ID factor ' | **NUM**

First(factor) = { (, ID , NUM }

26. factor ' \rightarrow [arithmetic_expression] | ϵ | (call '

First(factor ') = { [, ϵ , (}

27. call ' \rightarrow args) |)

First(call ') = {First(args)-{ ϵ }} \cup {) } = { (,) , ID , NUM }

28. args \rightarrow arithmetic_expression args_list '

First(args) = {First(arithmetic_expression)-{ ϵ }} \cup {First(args_list ')-{ ϵ }} = { (, ID , NUM , , , ϵ }

29. args_list ' \rightarrow , arithmetic_expression args_list ' | ϵ

First(args_list ') = { , , ϵ }

8 - Follow

Color Azul = First de Producciones

Color Amarillo = Follow de Producciones

1. $\text{program} \rightarrow \text{int ID declaration ' declaration_list ' | void ID (program '}$

$\text{First}(\text{program}) = \{ \text{int}, \text{void} \}$

$\text{Follow}(\text{program}) = \{ \$ \}$

2. $\text{program '} \rightarrow \text{param param_list ' } \{ \text{local_declarations statement_list } \} \text{ declaration_list ' | void) } \{ \text{local_declarations statement_list } \} \text{ program ''}$

$\text{First}(\text{program '}) = \{ \text{First}(\text{param}) - \{ \epsilon \} \} \cup \{ \text{void} \} = \{ \text{int}, \text{void} \}$

$\text{Follow}(\text{program '}) = \text{Follow}(\text{program '}) \cup \text{Follow}(\text{program}) = \{ \$ \}$

3. $\text{program ''} \rightarrow \text{declaration_list ' | } \epsilon$

$\text{First}(\text{program ''}) = \{ \text{First}(\text{declaration_list '}) - \{ \epsilon \} \} \cup \{ \epsilon \} = \{ \text{int}, \text{void}, \epsilon \}$

$\text{Follow}(\text{program ''}) = \text{Follow}(\text{program ''}) \cup \text{Follow}(\text{program '}) = \{ \$ \}$

4. $\text{declaration_list '} \rightarrow \text{int ID declaration ' declaration_list ' | void ID (params) } \{ \text{local_declarations statement_list } \} \text{ declaration_list ' | } \epsilon$

$\text{First}(\text{declaration_list '}) = \{ \text{int}, \text{void}, \epsilon \}$

$\text{Follow}(\text{declaration_list '}) = \text{Follow}(\text{program}) \cup \text{Follow}(\text{program '}) \cup \text{Follow}(\text{program ''}) \cup \text{Follow}(\text{declaration_list '}) \cup \text{Follow}(\text{declaration_list '}) = \{ \$ \}$

5. $\text{declaration '} \rightarrow ; | [\text{NUM}] ; | (\text{params}) \{ \text{local_declarations statement_list } \}$

$\text{First}(\text{declaration '}) = \{ ; , [, (\}$

$\text{Follow}(\text{declaration '}) = \text{First}(\text{declaration_list '}) - \{ \epsilon \} = \{ \text{int}, \text{void} \} \cup \text{Follow}(\text{program}) \cup \text{First}(\text{declaration_list '}) - \{ \epsilon \} = \{ \text{int}, \text{void} \} \cup \text{Follow}(\text{declaration_list '}) = \{ \text{int}, \text{void}, \$ \}$

6. $\text{var_declaration '} \rightarrow ; | [\text{NUM}] ;$

$\text{First}(\text{var_declaration '}) = \{ ; , [\}$

$\text{Follow}(\text{var_declaration '}) = \text{First}(\text{local_declarations}) - \{ \epsilon \} = \{ \text{int} \} \cup \text{Follow}(\text{local_declarations}) = \{ \text{ID}, \{ , \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \} , \text{int} \}$

7. $\text{params} \rightarrow \text{param param_list ' | void}$

$\text{First}(\text{params}) = \{\text{First}(\text{param}) - \{\epsilon\}\} \cup \{\text{void}\} = \{\text{int}, \text{void}\}$

$\text{Follow}(\text{params}) = \text{First}()) = \{\})$

8. $\text{param_list} \rightarrow , \text{param param_list} \mid \epsilon$

$\text{First}(\text{param_list}) = \{, , \epsilon\}$

$\text{Follow}(\text{param_list}) = \text{First}()) = \{\}) \cup \text{Follow}(\text{params}) \cup \text{Follow}(\text{param_list}) = \{\})$

9. $\text{param} \rightarrow \text{int ID param}$

$\text{First}(\text{param}) = \{\text{int}\}$

$\text{Follow}(\text{param}) = \text{First}(\text{param_list}) - \{\epsilon\} = \{, \} \cup \{, \} \cup \text{Follow}(\text{param_list}) = \{, \}$

$\text{Follow}(\text{params}) \text{First}()) = \{\}, \}$

10. $\text{param} \rightarrow \epsilon \mid []$

$\text{First}(\text{param}) = \{[, \epsilon\}$

$\text{Follow}(\text{param}) = \text{Follow}(\text{param}) \cup \text{Follow}(\text{param}) = \{\}, \}$

11. $\text{local_declarations} \rightarrow \text{int ID var_declaration local_declarations} \mid \epsilon$

$\text{First}(\text{local_declarations}) = \{\text{int}, \epsilon\}$

$\text{Follow}(\text{local_declarations}) = \text{First}(\text{statement_list}) - \{\epsilon\} = \{\text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}\} \cup \text{First}()) \cup \text{Follow}(\text{local_declarations}) = \{\text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \}$

12. $\text{statement_list} \rightarrow \text{ID statement local_declarations statement_list} \mid \{\text{local_declarations statement_list} \mid \text{if (expression) statement selection_stmt statement_list} \mid \text{while (expression) statement statement_list} \mid \text{return return_stmt statement_list} \mid \text{input ID var} ; \text{statement_list} \mid \text{output expression} ; \text{statement_list} \mid \epsilon$

$\text{First}(\text{statement_list}) = \{\text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \epsilon\}$

$\text{Follow}(\text{statement_list}) = \text{Follow}(\text{statement_list}) \cup \text{First}()) = \{\})$

13. $\text{statement} \rightarrow [\text{arithmetic_expression}] = \text{expression} ; \mid = \text{expression} ; \mid (\text{call} ;$

$\text{First}(\text{statement}) = \{[, =, (\}$

$\text{Follow}(\text{statement}) = \text{Follow}(\text{statement}) \cup \text{Follow}(\text{statement}) \cup \text{First}(\text{statement_list}) - \{\epsilon\} = \{\text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}\} \cup \text{Follow}(\text{statement_list}) = \{\text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \}, \text{else}\}$

14. statement \rightarrow ID statement ' | { local_declarations statement_list } | if (expression) statement selection_stmt ' | while (expression) statement | return return_stmt ' | input ID var ' ; | output expression ;

$\text{First}(\text{statement}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output} \}$

$\text{Follow}(\text{statement}) = \text{Follow}(\text{selection_stmt}) \cup \text{Follow}(\text{statement}) \cup \text{First}(\text{selection_stmt}) - \{ \epsilon \}$
 $= \{ \text{else} \} \cup \text{First}(\text{statement_list}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output} \} \cup$
 $\text{Follow}(\text{statement_list}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \}, \text{else} \}$

15. var ' \rightarrow [arithmetic_expression] | ϵ

$\text{First}(\text{var}') = \{ [, \epsilon \}$

$\text{Follow}(\text{var}') = \{ ; \}$

16. selection_stmt ' \rightarrow else statement | ϵ

$\text{First}(\text{selection_stmt}') = \{ \text{else}, \epsilon \}$

$\text{Follow}(\text{selection_stmt}') = \text{Follow}(\text{statement}) \cup \text{Follow}(\text{selection_stmt}') \cup \text{First}(\text{statement_list}) -$
 $\{ \epsilon \} = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output} \} \cup \text{Follow}(\text{statement_list}) = \{ \text{ID}, \{, \text{if},$
 $\text{while}, \text{return}, \text{input}, \text{output}, \}, \text{else} \}$

17. return_stmt ' \rightarrow ; | arithmetic_expression expression ' ;

$\text{First}(\text{return_stmt}') = \{ ; \} \cup \{ \text{First}(\text{arithmetic_expression}) - \{ \epsilon \} \} = \{ (, \text{ID}, \text{NUM}, ; \}$

$\text{Follow}(\text{return_stmt}') = \text{Follow}(\text{statement}) \cup \text{Follow}(\text{return_stmt}') \cup \text{First}(\text{statement_list}) - \{ \epsilon \} \cup$
 $\text{Follow}(\text{statement_list}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \}, \text{else} \}$

18. expression \rightarrow arithmetic_expression expression ' ;

$\text{First}(\text{expression}) = \{ \text{First}(\text{arithmetic_expression}) - \{ \epsilon \} \} = \{ (, \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{expression}) = \{ ;,) \}$

19. expression ' \rightarrow <= arithmetic_expression | < arithmetic_expression | > arithmetic_expression | >= arithmetic_expression | == arithmetic_expression | != arithmetic_expression | ϵ

$\text{First}(\text{expression}') = \{ <=, <, >, >=, ==, !=, \epsilon \}$

$\text{Follow}(\text{expression}') = \text{Follow}(\text{expression}) \cup \text{Follow}(\text{expression}') = \{ ;,) \}$

20. arithmetic_expression \rightarrow (arithmetic_expression) term ' arithmetic_expression ' | ID factor ' term ' arithmetic_expression ' | NUM term ' arithmetic_expression ' ;

$\text{First}(\text{arithmetic_expression}) = \{ (, \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{arithmetic_expression}) \cup \text{Follow}(\text{expression}') \cup \text{First}(\text{expression}') - \{ \epsilon \} = \{ <=, <, >, >=, ==, != \} \cup \text{Follow}(\text{expression}) = \{ ;,), <=, <, >, >=, ==, !=, ,,] \}$

21. $\text{arithmetic_expression}' \rightarrow + \text{term arithmetic_expression}' \mid - \text{term arithmetic_expression}' \mid \epsilon$

$\text{First}(\text{arithmetic_expression}') = \{ +, -, \epsilon \}$

$\text{Follow}(\text{arithmetic_expression}') = \text{Follow}(\text{arithmetic_expression}') \cup \text{Follow}(\text{arithmetic_expression}) = \{ ;,), <=, <, >, >=, ==, !=, ,,] \}$

22. $\text{term} \rightarrow (\text{arithmetic_expression}) \text{term}' \mid \text{ID factor}' \text{term}' \mid \text{NUM term}'$

$\text{First}(\text{term}) = \{ (, \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{term}) = \text{First}(\text{arithmetic_expression}') - \{ \epsilon \} = \{ +, - \} \cup \text{Follow}(\text{arithmetic_expression}') = \{ ;,), <=, <, >, >=, ==, !=, ,,] , +, - \}$

23. $\text{term}' \rightarrow \text{mulop factor term}' \mid \epsilon$

$\text{First}(\text{term}') = \{ \text{First}(\text{mulop}) - \{ \epsilon \} \} \cup \{ \epsilon \} = \{ *, /, \epsilon \}$

$\text{Follow}(\text{term}') = \text{Follow}(\text{term}) \cup \text{First}(\text{arithmetic_expression}') - \{ \epsilon \} = \{ +, - \} \cup \text{Follow}(\text{term}') \cup \text{Follow}(\text{arithmetic_expression}) = \{ ;,), <=, <, >, >=, ==, !=, ,,] , +, - \}$

24. $\text{mulop} \rightarrow * \mid /$

$\text{First}(\text{mulop}) = \{ *, / \}$

$\text{Follow}(\text{mulop}) = \text{First}(\text{factor}) - \{ \epsilon \} = \{ (, \text{ID}, \text{NUM} \} = \{ (, \text{ID}, \text{NUM} \}$

25. $\text{factor} \rightarrow (\text{arithmetic_expression}) \mid \text{ID factor}' \mid \text{NUM}$

$\text{First}(\text{factor}) = \{ (, \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{factor}) = \text{First}(\text{term}') - \{ \epsilon \} = \{ *, / \} \cup \text{Follow}(\text{term}') = \{ ;,), <=, <, >, >=, ==, !=, ,,] , +, -, *, / \}$

26. $\text{factor}' \rightarrow [\text{arithmetic_expression}] \mid \epsilon \mid (\text{call}'$

$\text{First}(\text{factor}') = \{ [, \epsilon, (\}$

$\text{Follow}(\text{factor}') = \text{Follow}(\text{factor}) \cup \text{Follow}(\text{factor}') \cup \text{First}(\text{term}') = \{ *, / \} \cup \text{Follow}(\text{term}') = \{ ;,), <=, <, >, >=, ==, !=, ,,] , +, - \} \cup \text{First}(\text{arithmetic_expression}') = \{ +, - \} = \{ ;,), <=, <, >, >=, ==, !=, ,,] , +, -, *, / \}$

27. $\text{call}' \rightarrow \text{args}) \mid)$

$\text{First}(\text{call } ') = \{\text{First}(\text{args}) - \{\epsilon\}\} \cup \{ \} \cup \{ \} = \{ (, , \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{call } ') = \text{Follow}(\text{factor } ') \cup \text{Follow}(\text{call } ') \cup \text{First}(;) = \{ ;, , , <=, <, >, >=, ==, !=, ,, ,], +, -, *, / \}$

28. $\text{args} \rightarrow \text{arithmetic_expression args_list } '$

$\text{First}(\text{args}) = \{\text{First}(\text{arithmetic_expression}) - \{\epsilon\}\} = \{ (, \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{args}) = \{ \} \}$

29. $\text{args_list } ' \rightarrow , \text{ arithmetic_expression args_list } ' \mid \epsilon$

$\text{First}(\text{args_list } ') = \{ , , \epsilon \}$

$\text{Follow}(\text{args_list } ') = \text{Follow}(\text{args_list } ') \cup \text{Follow}(\text{args}) = \{ \} \}$

8 - First +

Color Verde = First y Follow de Producciones

Color Amarillo = First+ de Producciones

Color Morado = Producciones Inútiles

Color Rojo = Problema del 'else'

1. $\text{program} \rightarrow \text{int ID declaration } ' \text{ declaration_list } ' \mid \text{void ID (program } '$

$\text{First}(\text{program}) = \{ \text{int}, \text{void} \}$

$\text{Follow}(\text{program}) = \{ \$ \}$

$\text{First}+(\text{program} \rightarrow \text{int ID declaration } ' \text{ declaration_list } ') = \{ \text{int} \}$

$\text{First}+(\text{program} \rightarrow \text{void ID (program } ') = \{ \text{void} \}$

2. $\text{program } ' \rightarrow \text{param param_list } ' \{ \text{local_declarations statement_list } \} \text{ declaration_list } ' \mid \text{void} \{ \text{local_declarations statement_list } \} \text{ declaration_list } '$

$\text{First}(\text{program } ') = \{\text{First}(\text{param}) - \{\epsilon\}\} \cup \{ \text{void} \} = \{ \text{int}, \text{void} \}$

$\text{Follow}(\text{program } ') = \text{Follow}(\text{program } ') \cup \text{Follow}(\text{program}) = \{ \$ \}$

$\text{First}+(\text{program } ' \rightarrow \text{param param_list } ' \{ \text{local_declarations statement_list } \} \text{ declaration_list } ') =$

$\text{First}(\text{param}) = \{ \text{int} \}$

First+(program ' --> **void**) { local_declarations statement_list } program ') =

{ **void** }

3. program ' --> declaration_list ' | ϵ

program ' --> **int ID** declaration ' declaration_list ' | **void ID** (params) { local_declarations statement_list } declaration_list ' | ϵ

Produccion inutil – Tiene lo mismo que declaration_list '

4. declaration_list ' --> **int ID** declaration ' declaration_list ' | **void ID** (params) { local_declarations statement_list } declaration_list ' | ϵ

First(declaration_list ') = { **int** , **void** , ϵ }

Follow(declaration_list ') = Follow(program) U Follow(program ') U Follow(program ") U Follow(declaration_list ') U Follow(declaration_list ') = { \$ }

First+(declaration_list ' --> **int ID** declaration ' declaration_list ') = { **int** }

First+(declaration_list ' --> **void ID** (params) { local_declarations statement_list } declaration_list ') = { **void** }

First+(declaration_list ' --> ϵ) = First(ϵ) U Follow(declaration_list ') = { ϵ , \$ }

5. declaration ' --> ; | [**NUM**] ; | (params) { local_declarations statement_list }

First(declaration ') = { ; , [, (}

Follow(declaration ') = First(declaration_list ') – { ϵ } = { **int** , **void** } U Follow(program) U First(declaration_list ') – { ϵ } = { **int** , **void** } U Follow(declaration_list ') = { **int** , **void** , \$ }

First+(declaration ' --> ;) = { ; }

First+(declaration ' --> [**NUM**] ;) = { [}

First+(declaration ' --> (params) { local_declarations statement_list }) = { (}

6. var_declaration ' --> ; | [**NUM**] ;

First(var_declaration ') = { ; , [}

Follow(var_declaration ') = First(local_declarations) – { ϵ } = { **int** } U Follow(local_declarations) = { **ID** , { , **if** , **while** , **return** , **input** , **output** , } , **int** }

First+(var_declaration ' --> ;) = { ; }

First+(var_declaration ' --> [NUM] ;) = { [] }

7. params --> param param_list ' | void

First(params) = {First(param) - { ϵ } } U { void } = { int , void }

Follow(params) = First() = { } }

First+(params --> param param_list ') = First(param) = { int }

First+(params --> void) = { void }

8. param_list ' --> , param param_list ' | ϵ

First(param_list ') = { , , ϵ }

Follow(param_list ') = First() = { } } U Follow(params) U Follow(param_list ') = { } }

First+(param_list ' --> , param param_list ') = { , }

First+(param_list ' --> ϵ) = First(ϵ) U Follow(param_list ') = { } , ϵ }

9. param --> int ID param '

First(param) = { int }

Follow(param) = First(param_list ') - { ϵ } = { , } U = { , } U Follow(param_list ') U Follow(params) First() = { } , , }

First+(param --> int ID param ') = { int }

10. param ' --> ϵ | []

First(param ') = { [, ϵ }

Follow(param ') = Follow(param ') U Follow(param) = { } , , }

First+(param ' --> ϵ) = First(ϵ) U Follow(param ') = { } , , , ϵ }

First+(param ' --> []) = { [] }

11. local_declarations --> int ID var_declaration ' local_declarations | ϵ

First(local_declarations) = { int , ϵ }

Follow(local_declarations) = First(statement_list) - { ϵ } = { ID , { , if , while , return , input , output } } U First() U Follow(local_declarations) = { ID , { , if , while , return , input , output , } }

$\text{First}^+(\text{local_declarations} \rightarrow \text{int ID var_declaration ' local_declarations}) = \{ \text{int} \}$

$\text{First}^+(\text{local_declarations} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{local_declarations}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \}, \epsilon \}$

12. $\text{statement_list} \rightarrow \text{ID statement ' statement_list} \mid \{ \text{local_declarations statement_list} \}$
 $\text{statement_list} \mid \text{if (expression) statement selection_stmt ' statement_list} \mid \text{while (}$
 $\text{expression) statement statement_list} \mid \text{return return_stmt ' statement_list} \mid \text{input ID var ' ;}$
 $\text{statement_list} \mid \text{output expression ; statement_list} \mid \epsilon$

$\text{First}(\text{statement_list}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \epsilon \}$

$\text{Follow}(\text{statement_list}) = \text{Follow}(\text{statement_list}) \cup \text{First}(\epsilon) = \{ \}$

$\text{First}^+(\text{statement_list} \rightarrow \text{ID statement ' statement_list}) = \{ \text{ID} \}$

$\text{First}^+(\text{statement_list} \rightarrow \{ \text{local_declarations statement_list} \} \text{statement_list}) = \{ \{ \}$

$\text{First}^+(\text{statement_list} \rightarrow \text{if (expression) statement selection_stmt ' statement_list}) = \{ \text{if} \}$

$\text{First}^+(\text{statement_list} \rightarrow \text{while (expression) statement statement_list}) = \{ \text{while} \}$

$\text{First}^+(\text{statement_list} \rightarrow \text{return return_stmt ' statement_list}) = \{ \text{return} \}$

$\text{First}^+(\text{statement_list} \rightarrow \text{input ID var ' ; statement_list}) = \{ \text{input} \}$

$\text{First}^+(\text{statement_list} \rightarrow \text{output expression ; statement_list}) = \{ \text{output} \}$

$\text{First}^+(\text{statement_list} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{statement_list}) = \{ \}, \epsilon \}$

13. $\text{statement ' } \rightarrow [\text{arithmetic_expression}] = \text{expression ;} \mid = \text{expression ;} \mid (\text{call ' ;}$

$\text{First}(\text{statement '}) = \{ [, =, (\}$

$\text{Follow}(\text{statement '}) = \text{Follow}(\text{statement}) \cup \text{Follow}(\text{statement '}) \cup \text{First}(\text{statement_list}) - \{ \epsilon \} = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output} \} \cup \text{Follow}(\text{statement_list}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \}, \text{else} \}$

$\text{First}^+(\text{statement ' } \rightarrow [\text{arithmetic_expression}] = \text{expression ;}) = \{ [\}$

$\text{First}^+(\text{statement ' } \rightarrow = \text{expression ;}) = \{ = \}$

$\text{First}^+(\text{statement ' } \rightarrow (\text{call ' ;}) = \{ (\}$

14. statement \rightarrow **ID** statement ' | { local_declarations statement_list } | **if** (expression) statement selection_stmt ' | **while** (expression) statement | **return** return_stmt ' | **input** ID var ' ; | **output** expression ;

$\text{First}(\text{statement}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output} \}$

$\text{Follow}(\text{statement}) = \text{Follow}(\text{selection_stmt}) \cup \text{Follow}(\text{statement}) \cup \text{First}(\text{selection_stmt}) - \{ \epsilon \}$
 $= \{ \text{else} \} \cup \text{First}(\text{statement_list}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output} \} \cup$
 $\text{Follow}(\text{statement_list}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \}, \text{else} \}$

$\text{First}^+(\text{statement} \rightarrow \text{ID statement}') = \{ \text{ID} \}$

$\text{First}^+(\text{statement} \rightarrow \{ \text{local_declarations statement_list} \}) = \{ \{ \}$

$\text{First}^+(\text{statement} \rightarrow \text{if (expression) statement selection_stmt}') = \{ \text{if} \}$

$\text{First}^+(\text{statement} \rightarrow \text{while (expression) statement}) = \{ \text{while} \}$

$\text{First}^+(\text{statement} \rightarrow \text{return return_stmt}') = \{ \text{return} \}$

$\text{First}^+(\text{statement} \rightarrow \text{input ID var}') = \{ \text{input} \}$

$\text{First}^+(\text{statement} \rightarrow \text{output expression ;}) = \{ \text{output} \}$

15. var ' \rightarrow [arithmetic_expression] | ϵ

$\text{First}(\text{var}') = \{ [, \epsilon \}$

$\text{Follow}(\text{var}') = \{ ; \}$

$\text{First}^+(\text{var}' \rightarrow [\text{arithmetic_expression}]) = \{ [\}$

$\text{First}^+(\text{var}' \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{var}') = \{ ;, \epsilon \}$

16. selection_stmt ' \rightarrow **else** statement | ϵ

$\text{First}(\text{selection_stmt}') = \{ \text{else}, \epsilon \}$

$\text{Follow}(\text{selection_stmt}') = \text{Follow}(\text{statement}) \cup \text{Follow}(\text{selection_stmt}') \cup \text{First}(\text{statement_list}) -$
 $\{ \epsilon \} = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output} \} \cup \text{Follow}(\text{statement_list}) = \{ \text{ID}, \{, \text{if},$
 $\text{while}, \text{return}, \text{input}, \text{output}, \}, \text{else} \}$

$\text{First}^+(\text{selection_stmt}' \rightarrow \text{else statement}) = \{ \text{else} \}$

$\text{First}^+(\text{selection_stmt}' \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{selection_stmt}') = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return},$
 $\text{input}, \text{output}, \}, \text{else}, \epsilon \}$

17. return_stmt ' \rightarrow ; | arithmetic_expression expression ' ;

$\text{First}(\text{return_stmt}) = \{ ; \} \cup \{ \text{First}(\text{arithmetic_expression}) - \{ \epsilon \} \} \cup \{ \text{First}(\text{expression}) - \{ \epsilon \} \} = \{ (, \text{ID}, \text{NUM}, ;, <=, <, >, >=, ==, !=, \epsilon \}$

$\text{Follow}(\text{return_stmt}) = \text{Follow}(\text{statement}) \cup \text{Follow}(\text{return_stmt}) \cup \text{First}(\text{statement_list} - \{ \epsilon \}) \cup \text{Follow}(\text{statement_list}) = \{ \text{ID}, \{, \text{if}, \text{while}, \text{return}, \text{input}, \text{output}, \}, \text{else} \}$

$\text{First}+(\text{return_stmt} \rightarrow ;) = \{ ; \}$

$\text{First}+(\text{return_stmt} \rightarrow \text{arithmetic_expression expression} ;) = \text{First}(\text{arithmetic_expression}) = \{ (, \text{ID}, \text{NUM} \}$

18. $\text{expression} \rightarrow \text{arithmetic_expression expression}$

$\text{First}(\text{expression}) = \{ \text{First}(\text{arithmetic_expression}) - \{ \epsilon \} \} \cup \{ \text{First}(\text{expression}) - \{ \epsilon \} \} = \{ (, \text{ID}, \text{NUM}, <=, <, >, >=, ==, !=, \epsilon \}$

$\text{Follow}(\text{expression}) = \{ ;,) \}$

$\text{First}+(\text{expression} \rightarrow \text{arithmetic_expression expression}) = \text{First}(\text{arithmetic_expression}) = \{ (, \text{ID}, \text{NUM} \}$

19. $\text{expression} \rightarrow <= \text{arithmetic_expression} \mid < \text{arithmetic_expression} \mid > \text{arithmetic_expression} \mid >= \text{arithmetic_expression} \mid == \text{arithmetic_expression} \mid != \text{arithmetic_expression} \mid \epsilon$

$\text{First}(\text{expression}) = \{ <=, <, >, >=, ==, !=, \epsilon \}$

$\text{Follow}(\text{expression}) = \text{Follow}(\text{expression}) \cup \text{Follow}(\text{expression}) = \{ ;,) \}$

$\text{First}+(\text{expression} \rightarrow <= \text{arithmetic_expression}) = \{ <= \}$

$\text{First}+(\text{expression} \rightarrow < \text{arithmetic_expression}) = \{ < \}$

$\text{First}+(\text{expression} \rightarrow > \text{arithmetic_expression}) = \{ > \}$

$\text{First}+(\text{expression} \rightarrow >= \text{arithmetic_expression}) = \{ >= \}$

$\text{First}+(\text{expression} \rightarrow == \text{arithmetic_expression}) = \{ == \}$

$\text{First}+(\text{expression} \rightarrow != \text{arithmetic_expression}) = \{ != \}$

$\text{First}+(\text{expression} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{expression}) = \{ ;,) , \epsilon \}$

20. $\text{arithmetic_expression} \rightarrow (\text{arithmetic_expression}) \text{term} \mid \text{arithmetic_expression} \mid \text{ID factor} \mid \text{term} \mid \text{arithmetic_expression} \mid \text{NUM term} \mid \text{arithmetic_expression}$

$\text{First}(\text{arithmetic_expression}) = \{ (, \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{arithmetic_expression}) = \{ \} , \} \} \cup \text{Follow}(\text{expression '}) \cup \text{First}(\text{expression '}) - \{ \epsilon \} = \{ <= , < , > , >= , == , != \} \cup \text{Follow}(\text{expression}) = \{ ; ,) , <= , < , > , >= , == , != , , ,] \}$

$\text{First}+(\text{arithmetic_expression} \rightarrow (\text{arithmetic_expression}) \text{ term ' arithmetic_expression '}) = \{ (\}$

$\text{First}+(\text{arithmetic_expression} \rightarrow \text{ID factor ' term ' arithmetic_expression '}) = \{ \text{ID} \}$

$\text{First}+(\text{arithmetic_expression} \rightarrow \text{NUM term ' arithmetic_expression '}) = \{ \text{NUM} \}$

21. $\text{arithmetic_expression '} \rightarrow + \text{ term arithmetic_expression '} \mid - \text{ term arithmetic_expression '} \mid \epsilon$

$\text{First}(\text{arithmetic_expression '}) = \{ + , - , \epsilon \}$

$\text{Follow}(\text{arithmetic_expression '}) = \text{Follow}(\text{arithmetic_expression '}) \cup \text{Follow}(\text{arithmetic_expression}) = \{ ; ,) , <= , < , > , >= , == , != , , ,] \}$

$\text{First}+(\text{arithmetic_expression '} \rightarrow + \text{ term arithmetic_expression '}) = \{ + \}$

$\text{First}+(\text{arithmetic_expression '} \rightarrow - \text{ term arithmetic_expression '}) = \{ - \}$

$\text{First}+(\text{arithmetic_expression '} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{arithmetic_expression '}) = \{ ; ,) , <= , < , > , >= , == , != , , ,] , \epsilon \}$

22. $\text{term} \rightarrow (\text{arithmetic_expression}) \text{ term '} \mid \text{ID factor ' term '} \mid \text{NUM term '}$

$\text{First}(\text{term}) = \{ (, \text{ID} , \text{NUM} \}$

$\text{Follow}(\text{term}) = \text{First}(\text{arithmetic_expression '}) - \{ \epsilon \} = \{ + , - \} \cup \text{Follow}(\text{arithmetic_expression '}) = \{ ; ,) , <= , < , > , >= , == , != , , ,] , + , - \}$

$\text{First}+(\text{term} \rightarrow (\text{arithmetic_expression}) \text{ term '}) = \{ (\}$

$\text{First}+(\text{term} \rightarrow \text{ID factor ' term '}) = \{ \text{ID} \}$

$\text{First}+(\text{term} \rightarrow \text{NUM term '}) = \{ \text{NUM} \}$

23. $\text{term '} \rightarrow \text{mulop factor term '} \mid \epsilon$

$\text{First}(\text{term '}) = \{ \text{First}(\text{mulop}) - \{ \epsilon \} \} \cup \{ \text{First}(\text{factor}) - \{ \epsilon \} \} \cup \{ \text{First}(\text{term '}) - \{ \epsilon \} \} \cup \{ \epsilon \} = \{ * , / , \epsilon , (, \text{ID} , \text{NUM} \}$

$\text{Follow}(\text{term '}) = \text{Follow}(\text{term}) \cup \text{First}(\text{arithmetic_expression '}) - \{ \epsilon \} = \{ + , - \} \cup \text{Follow}(\text{term '}) \cup \text{Follow}(\text{arithmetic_expression}) = \{ ; ,) , <= , < , > , >= , == , != , , ,] , + , - \}$

$\text{First}+(\text{term '} \rightarrow \text{mulop factor term '}) = \text{First}(\text{mulop}) = \{ * , / \}$

$\text{First}+(\text{term '} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{term '}) = \{ ; ,) , <= , < , > , >= , == , != , , ,] , + , - , \epsilon \}$

24. $\text{mulop} \rightarrow * \mid /$

$\text{First}(\text{mulop}) = \{ *, / \}$

$\text{Follow}(\text{mulop}) = \text{First}(\text{factor}) - \{ \epsilon \} = \{ (, \text{ID}, \text{NUM} \} \cup \text{First}(\text{term}') - \{ \epsilon \} = \{ *, /, (, \text{ID}, \text{NUM} \}$

$= \{ *, /, (, \text{ID}, \text{NUM} \}$

$\text{First}^+(\text{mulop} \rightarrow *) = \{ * \}$

$\text{First}^+(\text{mulop} \rightarrow /) = \{ / \}$

25. $\text{factor} \rightarrow (\text{arithmetic_expression}) \mid \text{ID factor}' \mid \text{NUM}$

$\text{First}(\text{factor}) = \{ (, \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{factor}) = \text{First}(\text{term}') - \{ \epsilon \} = \{ *, /, (, \text{ID}, \text{NUM} \} \cup \text{Follow}(\text{term}') = \{ ;,), <=, <, >, >=, ==, !=, ,,], +, -, *, /, (, \text{ID}, \text{NUM} \}$

$\text{First}^+(\text{factor} \rightarrow (\text{arithmetic_expression})) = \{ (\}$

$\text{First}^+(\text{factor} \rightarrow \text{ID factor}') = \{ \text{ID} \}$

$\text{First}^+(\text{factor} \rightarrow \text{NUM}) = \{ \text{NUM} \}$

26. $\text{factor}' \rightarrow [\text{arithmetic_expression}] \mid \epsilon \mid (\text{call}'$

$\text{First}(\text{factor}') = \{ [, \epsilon, (\}$

$\text{Follow}(\text{factor}') = \text{Follow}(\text{factor}) \cup \text{Follow}(\text{factor}') \cup \text{First}(\text{term}') = \{ *, /, (, \text{ID}, \text{NUM} \} \cup$

$\text{Follow}(\text{term}) \cup \text{First}(\text{arithmetic_expression}') = \{ +, - \} = \{ ;,), <=, <, >, >=, ==, !=, ,,], +, -, *, /, (, \text{ID}, \text{NUM} \}$

$\text{First}^+(\text{factor}' \rightarrow [\text{arithmetic_expression}]) = \{ [\}$

$\text{First}^+(\text{factor}' \rightarrow (\text{call}')) = \{ (\}$

$\text{First}^+(\text{factor}' \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{factor}') = \{ +, - \} = \{ ;,), <=, <, >, >=, ==, !=, ,,], +, -, *, / \}$

27. $\text{call}' \rightarrow \text{args}) \mid)$

$\text{First}(\text{call}') = \{ \text{First}(\text{args}) - \{ \epsilon \} \} \cup \{) \} = \{ (,), \text{ID}, \text{NUM} \}$

$\text{Follow}(\text{call}') = \text{Follow}(\text{factor}') \cup \text{Follow}(\text{call}') \cup \text{First}(;) = \{ ;,), <=, <, >, >=, ==, !=, ,,], +, -, *, /, (, \text{ID}, \text{NUM} \}$

$\text{First}^+(\text{call}' \rightarrow \text{args}) = \text{First}(\text{args}) = \{ (, \text{ID}, \text{NUM}, ,, \epsilon \} = \{ (, \text{ID}, \text{NUM}, ,, \epsilon \}$

$\text{First}+(\text{call } ' \rightarrow) = \{ \} \}$

28. $\text{args } \rightarrow \text{arithmetic_expression args_list } '$

$\text{First}(\text{args}) = \{\text{First}(\text{arithmetic_expression}) - \{ \epsilon \} \} \cup \{\text{First}(\text{args_list } ') - \{ \epsilon \} \} = \{ (, \text{ID}, \text{NUM}, , , \epsilon \}$

$\text{Follow}(\text{args}) = \{) \}$

$\text{First}+(\text{args } \rightarrow \text{arithmetic_expression args_list } ') = \text{First}(\text{arithmetic_expression}) = \{ (, \text{ID}, \text{NUM} \}$

29. $\text{args_list } ' \rightarrow , \text{arithmetic_expression args_list } ' \mid \epsilon$

$\text{First}(\text{args_list } ') = \{ , , \epsilon \}$

$\text{Follow}(\text{args_list } ') = \text{Follow}(\text{args_list } ') \cup \text{Follow}(\text{args}) = \{) \}$

$\text{First}+(\text{args_list } ' \rightarrow , \text{arithmetic_expression args_list } ') = \{ , \}$

$\text{First}+(\text{args_list } ' \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{args_list } ') = \{) , \epsilon \}$

9 – Producciones Finales

1. $\text{First}+(\text{program } \rightarrow \text{int ID declaration } ' \text{ declaration_list } ') = \{ \text{int} \}$

2. $\text{First}+(\text{program } \rightarrow \text{void ID (program } ') = \{ \text{void} \}$

3. $\text{First}+(\text{program } ' \rightarrow \text{param param_list } ') \{ \text{local_declarations statement_list } \}$
 $\text{declaration_list } ') =$

$\text{First}(\text{param}) = \{ \text{int} \}$

4. $\text{First}+(\text{program } ' \rightarrow \text{void}) \{ \text{local_declarations statement_list } \} \text{ declaration_list } ') =$
 $\{ \text{void} \}$

5. $\text{First}+(\text{declaration_list } ' \rightarrow \text{int ID declaration } ' \text{ declaration_list } ') = \{ \text{int} \}$

6. $\text{First}+(\text{declaration_list } ' \rightarrow \text{void ID (params) } \{ \text{local_declarations statement_list } \}$
 $\text{declaration_list } ') = \{ \text{void} \}$

7. $\text{First}+(\text{declaration_list } ' \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{declaration_list } ') = \{ \epsilon , \$ \}$

8. $\text{First}+(\text{declaration } ' \rightarrow ;) = \{ ; \}$

9. $\text{First}+(\text{declaration } ' \rightarrow [\text{NUM}] ;) = \{ [\}$

10. $\text{First}+(\text{declaration } ' \rightarrow (\text{ params) } \{ \text{local_declarations statement_list } \}) = \{ (\}$

11. $\text{First}+(\text{var_declaration } ' \rightarrow ;) = \{ ; \}$
12. $\text{First}+(\text{var_declaration } ' \rightarrow [\text{ NUM }] ;) = \{ [\}$
13. $\text{First}+(\text{params } \rightarrow \text{ param param_list } ') = \text{First}(\text{param}) = \{ \text{int} \}$
14. $\text{First}+(\text{params } \rightarrow \text{ void }) = \{ \text{void} \}$
15. $\text{First}+(\text{param_list } ' \rightarrow , \text{ param param_list } ') = \{ , \}$
16. $\text{First}+(\text{ param_list } ' \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{param_list } ') = \{) , \epsilon \}$
17. $\text{First}+(\text{param } \rightarrow \text{ int ID param } ') = \{ \text{int} \}$
18. $\text{First}+(\text{param } ' \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{param } ') = \{) , , \epsilon \}$
19. $\text{First}+(\text{param } ' \rightarrow []) = \{ [\}$
20. $\text{First}+(\text{local_declarations } \rightarrow \text{ int ID var_declaration } ' \text{ local_declarations}) = \{ \text{int} \}$
21. $\text{First}+(\text{local_declarations } \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{local_declarations}) = \{ \text{ID} , \{ , \text{if} , \text{while} , \text{return} , \text{input} , \text{output} , \} , \epsilon \}$
22. $\text{First}+(\text{statement_list } \rightarrow \text{ ID statement } ' \text{ statement_list }) = \{ \text{ID} \}$
23. $\text{First}+(\text{statement_list } \rightarrow \{ \text{ local_declarations statement_list } \} \text{ statement_list }) = \{ \{ \}$
24. $\text{First}+(\text{statement_list } \rightarrow \text{ if } (\text{ expression }) \text{ statement selection_stmt } ' \text{ statement_list }) = \{ \text{if} \}$
25. $\text{First}+(\text{statement_list } \rightarrow \text{ while } (\text{ expression }) \text{ statement statement_list }) = \{ \text{while} \}$
26. $\text{First}+(\text{statement_list } \rightarrow \text{ return return_stmt } ' \text{ statement_list }) = \{ \text{return} \}$
27. $\text{First}+(\text{statement_list } \rightarrow \text{ input ID var } ' ; \text{ statement_list }) = \{ \text{input} \}$
28. $\text{First}+(\text{statement_list } \rightarrow \text{ output expression } ; \text{ statement_list }) = \{ \text{output} \}$
29. $\text{First}+(\text{statement_list } \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{statement_list}) = \{ \} , \epsilon \}$
30. $\text{First}+(\text{statement } ' \rightarrow [\text{ arithmetic_expression }] = \text{ expression } ;) = \{ [\}$
31. $\text{First}+(\text{statement } ' \rightarrow = \text{ expression } ;) = \{ = \}$
32. $\text{First}+(\text{statement } ' \rightarrow (\text{ call } ' ;) = \{ (\}$
33. $\text{First}+(\text{statement } \rightarrow \text{ ID statement } ') = \{ \text{ID} \}$

34. $\text{First}^+(\text{statement} \rightarrow \{ \text{local_declarations statement_list} \}) = \{ \{ \}$
35. $\text{First}^+(\text{statement} \rightarrow \text{if (expression) statement selection_stmt '}) = \{ \text{if} \}$
36. $\text{First}^+(\text{statement} \rightarrow \text{while (expression) statement}) = \{ \text{while} \}$
37. $\text{First}^+(\text{statement} \rightarrow \text{return return_stmt '}) = \{ \text{return} \}$
38. $\text{First}^+(\text{statement} \rightarrow \text{input ID var ' ;}) = \{ \text{input} \}$
39. $\text{First}^+(\text{statement} \rightarrow \text{output expression ;}) = \{ \text{output} \}$
40. $\text{First}^+(\text{var '} \rightarrow [\text{arithmetic_expression}]) = \{ [] \}$
41. $\text{First}^+(\text{var '} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{var '}) = \{ ; , \epsilon \}$
42. $\text{First}^+(\text{selection_stmt '} \rightarrow \text{else statement}) = \{ \text{else} \}$
43. $\text{First}^+(\text{selection_stmt '} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{selection_stmt '}) = \{ \text{ID} , \{ , \text{if} , \text{while} , \text{return} , \text{input} , \text{output} , \} , \text{else} , \epsilon \}$
44. $\text{First}^+(\text{return_stmt '} \rightarrow ;) = \{ ; \}$
45. $\text{First}^+(\text{return_stmt '} \rightarrow \text{arithmetic_expression expression ' ;}) = \text{First}(\text{arithmetic_expression}) = \{ (, \text{ID} , \text{NUM} \}$
46. $\text{First}^+(\text{expression} \rightarrow \text{arithmetic_expression expression '}) = \text{First}(\text{arithmetic_expression}) = \{ (, \text{ID} , \text{NUM} \}$
47. $\text{First}^+(\text{expression '} \rightarrow \leq \text{arithmetic_expression}) = \{ \leq \}$
48. $\text{First}^+(\text{expression '} \rightarrow < \text{arithmetic_expression}) = \{ < \}$
49. $\text{First}^+(\text{expression '} \rightarrow > \text{arithmetic_expression}) = \{ > \}$
50. $\text{First}^+(\text{expression '} \rightarrow \geq \text{arithmetic_expression}) = \{ \geq \}$
51. $\text{First}^+(\text{expression '} \rightarrow == \text{arithmetic_expression}) = \{ == \}$
52. $\text{First}^+(\text{expression '} \rightarrow != \text{arithmetic_expression}) = \{ != \}$
53. $\text{First}^+(\text{expression '} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{expression '}) = \{ ; ,) , \epsilon \}$
54. $\text{First}^+(\text{arithmetic_expression} \rightarrow (\text{arithmetic_expression}) \text{term ' arithmetic_expression '}) = \{ (\}$
55. $\text{First}^+(\text{arithmetic_expression} \rightarrow \text{ID factor ' term ' arithmetic_expression '}) = \{ \text{ID} \}$

56. $\text{First}+(\text{arithmetic_expression} \rightarrow \text{NUM term ' arithmetic_expression '}) = \{ \text{NUM} \}$
57. $\text{First}+(\text{arithmetic_expression} \rightarrow + \text{ term arithmetic_expression '}) = \{ + \}$
58. $\text{First}+(\text{arithmetic_expression} \rightarrow - \text{ term arithmetic_expression '}) = \{ - \}$
59. $\text{First}+(\text{arithmetic_expression} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{arithmetic_expression '}) = \{ ; , , , \leq , < , > , \geq , == , != , , ,] , + , - , \epsilon \}$
60. $\text{First}+(\text{term} \rightarrow (\text{ arithmetic_expression }) \text{ term '}) = \{ (\}$
61. $\text{First}+(\text{term} \rightarrow \text{ID factor ' term '}) = \{ \text{ID} \}$
62. $\text{First}+(\text{term} \rightarrow \text{NUM term '}) = \{ \text{NUM} \}$
63. $\text{First}+(\text{term '} \rightarrow \text{mulop factor term '}) = \text{First}(\text{mulop}) = \{ * , / \}$
64. $\text{First}+(\text{term '} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{term '}) = \{ ; , , , \leq , < , > , \geq , == , != , , ,] , + , - , \epsilon \}$
65. $\text{First}+(\text{mulop} \rightarrow *) = \{ * \}$
66. $\text{First}+(\text{mulop} \rightarrow /) = \{ / \}$
67. $\text{First}+(\text{factor} \rightarrow (\text{ arithmetic_expression })) = \{ (\}$
68. $\text{First}+(\text{factor} \rightarrow \text{ID factor '}) = \{ \text{ID} \}$
69. $\text{First}+(\text{factor} \rightarrow \text{NUM}) = \{ \text{NUM} \}$
70. $\text{First}+(\text{factor '} \rightarrow [\text{ arithmetic_expression }]) = \{ [\}$
71. $\text{First}+(\text{factor '} \rightarrow (\text{ call '}) = \{ (\}$
72. $\text{First}+(\text{factor '} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{factor '}) = \{ + , - \} = \{ ; , , , \leq , < , > , \geq , == , != , , ,] , + , - , * , / \}$
73. $\text{First}+(\text{call '} \rightarrow \text{args})) = \text{First}(\text{args}) = \{ (, \text{ID} , \text{NUM} , , , \epsilon \} = \{ (, \text{ID} , \text{NUM} , , , \epsilon \}$
74. $\text{First}+(\text{call '} \rightarrow)) = \{) \}$
75. $\text{First}+(\text{args} \rightarrow \text{arithmetic_expression args_list '}) = \text{First}(\text{arithmetic_expression}) = \{ (, \text{ID} , \text{NUM} \}$
76. $\text{First}+(\text{args_list '} \rightarrow , \text{ arithmetic_expression args_list '}) = \{ , \}$
77. $\text{First}+(\text{args_list '} \rightarrow \epsilon) = \text{First}(\epsilon) \cup \text{Follow}(\text{args_list '}) = \{) , \epsilon \}$

2.3. Mensajes de error

Los mensajes de error se dividen en 4 diferentes secciones:

1- Errores en cada producción, a los cuales se llegaba si no existía un token con cual seguir en esa producción.

args_list_prime -> "Error args_list_prime en la linea "
 args -> "Error, argumentos mal formados en la linea "
 call_prime -> "Error, llamada mal formada en la linea "
 factor_prime -> "Error, ningun operador despues de 'Identificador' mal formado en la linea "
 factor -> "Error, factor mal formado en la linea "
 mulop -> "Error, '*' o '/' faltantes en la linea "
 term_prime -> "Error, operadores faltantes o termino de expression faltante(s) en la linea "
 term -> "Error, termino mal formado en la linea "
 arithmetic_expression_prime -> "Error, expresion aritmetica despues de suma o resta mal formada en la linea "
 arithmetic_expression -> "Error, expresion aritmetica mal formada en la linea "
 expression_prime -> "Error, mal uso de operadores en expression en la linea "
 expression -> "Error, expresion aritmetica mal formada en la linea "
 return_stmt_prime -> "Error, enunciado de return malformado en la linea "
 selection_stmt_prime -> "Error, enunciado de seleccion malformado en la linea "
 var_prime -> "Error, variable malformada en la linea "
 statement_prime -> "Error, variable o funcion definida/declarada incorrectamente en la linea "
 statement -> "Error, enunciado mal formado en la linea "
 statement_list -> "Error, Cerrar llaves en la linea "
 local_declarations -> "Error, declaracion de variable o funcion local mal formadas en la linea "
 param_prime -> "Error, braquetas mal formadas despues de parametros en la linea "
 param -> "Error, parametro mal formado en la linea "
 param_list_prime -> "Error, lista de parametros mal formada en la linea "
 params -> "Error, parametros mal establecidos en la linea "
 var_declaration_prime -> "Error, declaracion de variable mal formada en la linea "
 declaration_prime -> "Error, declaracion mal formada en la linea "
 declaration_list_prime -> "Error, declaracion de funcion incorrecta en la linea "
 program_prime -> "Error, programa despues de declaracion con 'void' malformado en la linea "
 program -> "Error, el programa debe ser inicializado con declaraciones 'int' o 'void' en la linea "

2- Errores si es que existía más de una función o variable global con el mismo nombre.

program -> "No pueden existir mas de una funcion con el mismo nombre, linea "
 declaration_list_prime -> "No pueden existir mas de una funcion con el mismo nombre, linea "
 declaration_prime -> "No pueden existir mas de una variable global con el mismo nombre , linea "
 "

3- Error en caso de que la última declaración no sea una función main.

declaration_list_prime -> "Error, favor de terminar con funcion 'main'"

4 - Error en caso de match.

match->

"Error, uso o mala colocacion de 'RETURN' en la linea "
 "Error, uso o mala colocacion de 'WHILE' en la linea "
 "Error, uso o mala colocacion de 'VOID' en la linea "
 "Error, uso o mala colocacion de 'ELSE' en la linea "
 "Error, uso o mala colocacion de 'IF' en la linea "
 "Error, uso o mala colocacion de 'INT' en la linea "
 "Error, uso o mala colocacion de 'INPUT' en la linea "
 "Error, uso o mala colocacion de 'OUTPUT' en la linea "
 "Error, uso o mala colocacion de '>' en la linea "
 "Error, uso o mala colocacion de '<' en la linea "
 "Error, uso o mala colocacion de '=' en la linea "
 "Error, uso o mala colocacion de '<=' en la linea "
 "Error, uso o mala colocacion de '==' en la linea "
 "Error, uso o mala colocacion de '>=' en la linea "
 "Error, uso o mala colocacion de '!=' en la linea "
 "Error, uso o mala colocacion de '(' en la linea "
 "Error, uso o mala colocacion de ')' en la linea "
 "Error, uso o mala colocacion de '[' en la linea "
 "Error, uso o mala colocacion de ']' en la linea "
 "Error, uso o mala colocacion de '{' en la linea "
 "Error, uso o mala colocacion de '}' en la linea "
 "Error, uso o mala colocacion de '/' en la linea "
 "Error, uso o mala colocacion de '*' en la linea "
 "Error, uso o mala colocacion de '-' en la linea "
 "Error, uso o mala colocacion de '+' en la linea "
 "Error, uso o mala colocacion de ',' en la linea "
 "Error, uso o mala colocacion de ';' en la linea "
 "Error, uso o mala colocacion de 'ID' en la linea "
 "Error, uso o mala colocacion de 'Int' en la linea "
 "Error Match en la linea "

3. Diseño

3.1. Pseudocódigo

```
// Match function
int Match(Token terminal) {
    if (current_token == terminal)
        current_token = Get_Next-Token();
    else
        Error();
} /* end match*/

// Main function
void main () {
    Token current_token;
    addToken("$");
    current_token = Get_Next-Token();
    exp();
    if (current_token == $) Syntax_Analysis_OK
    else Syntax_Analysis_Error
}
```

3.2. Tabla de Parsing

	ID	void	int	NUM	if	while	return	input	output	else	-	+	/	*	{	}	()	[]	;	,	>	<	=	<=	>=	==	!=	!	\$
program	Error	2	1	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
program '	Error	4	3	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
declaration_list '	Error	6	5	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	7
declaration '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	10	Error	9	Error	8	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
var_declaration '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	12	Error	11	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
params	Error	14	13	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
param_list '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	16	Error	Error	Error	15	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
param	Error	Error	17	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	18	19	Error	18	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
param '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
local_declarations	21	Error	20	Error	21	21	21	21	21	Error	Error	Error	Error	Error	21	21	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
statement_list	22	Error	Error	Error	24	25	26	27	28	Error	Error	Error	Error	Error	23	29	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
statement '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	32	Error	30	Error	Error	Error	Error	Error	Error	Error	Error	Error	31	Error	Error
statement	33	Error	Error	Error	35	36	37	38	39	Error	Error	Error	Error	Error	34	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
var '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	40	Error	41	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
selection_stmt '	43	Error	Error	Error	43	43	43	43	43	42, 43	Error	Error	Error	Error	Error	43	43	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
return_stmt '	45	Error	Error	45	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	45	Error	Error	44	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
expression	46	Error	Error	46	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	46	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
expression '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	53	Error	53	Error	49	48	Error	47	50	51	52	Error	Error	Error
arithmetic_expression	55	Error	Error	56	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	54	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
arithmetic_expression '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	58	57	Error	Error	Error	Error	Error	59	Error	59	59	59	59	59	Error	59	59	59	59	Error	Error
term	61	Error	Error	62	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	60	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
term '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	64	64	63	63	Error	Error	Error	64	Error	64	64	Error	64	64	Error	64	64	64	64	Error	Error
mulop	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	66	65	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
factor	68	Error	Error	69	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	67	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
factor '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	72	72	72	72	Error	Error	71	72	70	72	72	72	72	72	Error	72	72	72	72	Error	Error
call '	73	Error	Error	73	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	73	74	Error	Error	Error	73	Error	Error	Error	Error	Error	Error	Error	Error	Error
args	75	Error	Error	75	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	75	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
args_list '	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	77	Error	Error	Error	76	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error

Nota: La Tabla de Parsing se realizó para revisar donde había producciones que apuntaban a él mismo terminal los cuales daban conflicto.

El error del else se trabajó dentro del código ya que las producciones número 42 y número 43 este de repetía, por lo cual se decidió que la producción 42 es la que se conservaría el else.

3.3. Tablas de Símbolos

Entrada	Var Locales	Var Globales	Funcion	Llamadas a funcion	Numero de Parametros de funcion
Tokens 28,X	Numero de variables locales	Numero de variables globales	Numero de funciones	Llamadas de funcion dentro de otra	Numero de parametros en caso de funcion

Para la actualización de las tablas de símbolos solamente se utilizó la tabla de identificadores ya que esta tabla es la que realmente se actualiza a comparación de la tabla de constantes enteras, se modificó para que se pudieran identificar las llamadas a función, variables globales, variables locales y el número de parámetros de una función y así poder arrojarlas al usuario.

4. Implementación

4.1. Código Fuente

```
// current_token(Token Actual) y vecParPos(Posicion del vector parser) global para que se
pueda manejar facilmente
string current_token;
int vecParPos;

// Divisor de coma en tokens <28,x> y <29,x> en donde x es un valor entero sin afectar el token
actual
string split_c_forif(string str)
{
    if (current_token == "30") {
        cout << "Falta cerrar llaves de funcion en linea "<< vecLinea.getIdent(vecParPos) << endl;
        abort();
    }
    else if (current_token == "1" || current_token == "2" || current_token == "3" || current_token ==
"4" || current_token == "5" || current_token == "6" || current_token == "7" ||
current_token == "8" || current_token == "9" || current_token == "10" || current_token ==
"11" || current_token == "12" || current_token == "13" || current_token == "14" ||
current_token == "15" || current_token == "16" || current_token == "17" || current_token ==
"18" || current_token == "19" || current_token == "20" || current_token == "21" ||
current_token == "22" || current_token == "23" || current_token == "24" || current_token ==
"25" || current_token == "26" || current_token == "27") {
        return current_token;
    }
    else {
        string a = "";
        for (auto b : str)
        {
            if (b == ',')
            {
                return a;
            }
        }
        else {
```



```

        a = a + b;
    }
}
}

```

// Divisor de coma en tokens <28,x> y <29,x> en donde x es un valor entero afectando el token actual

```

void split_c(string str)
{
    string a = "";
    for (auto b : str)
    {
        if (b == ',')
        {
            current_token = a;
            return;
        }
        else {
            a = a + b;
        }
    }
}

```

// Funciones para llamar antes de la ejecución

```

void arithmetic_expression();
void local_declarations();
void statement();
void statement_list();
void var_declaration_prime();

```

// Funcion de match para avanzar de token

```

void match(string terminal) {
    if (current_token == terminal) {
        vecParPos++;
        current_token = vecOutput.getIdent(vecParPos);
        //current_token = Get_Next-Token();
    }
    else {
        //Error();
        if (current_token == "1") {

            cout << "Error, uso o mala colocacion de 'RETURN' en la linea " <<
            vecLinea.getIdent(vecParPos + 2) << endl;

```

```

        abort();
    }
    else if (current_token == "2") {

        cout << "Error, uso o mala colocacion de 'WHILE' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "3") {

        cout << "Error, uso o mala colocacion de 'VOID' en la linea " <<
vecLinea.getIdent(vecParPos ) << endl;
        abort();
    }
    else if (current_token == "4") {

        cout << "Error, uso o mala colocacion de 'ELSE' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "5") {

        cout << "Error, uso o mala colocacion de 'IF' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "6") {

        cout << "Error, uso o mala colocacion de 'INT' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "7") {

        cout << "Error, uso o mala colocacion de 'INPUT' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "8") {

        cout << "Error, uso o mala colocacion de 'OUTPUT' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

```

```

else if (current_token == "9") {

    cout << "Error, uso o mala colocacion de '>' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
else if (current_token == "10") {

    cout << "Error, uso o mala colocacion de '<' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
else if (current_token == "11") {

    cout << "Error, uso o mala colocacion de '=' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
else if (current_token == "12") {

    cout << "Error, uso o mala colocacion de '<=' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
else if (current_token == "13") {

    cout << "Error, uso o mala colocacion de '==' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
else if (current_token == "14") {

    cout << "Error, uso o mala colocacion de '>=' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
else if (current_token == "15") {

    cout << "Error, uso o mala colocacion de '!=' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
else if (current_token == "16") {

```

```

        cout << "Error, uso o mala colocacion de '(' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "17") {

        cout << "Error, uso o mala colocacion de ')' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "18") {

        cout << "Error, uso o mala colocacion de '[' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "19") {

        cout << "Error, uso o mala colocacion de ']' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "20") {

        cout << "Error, uso o mala colocacion de '{' en la linea " <<
vecLinea.getIdent(vecParPos) << endl;
        abort();
    }
    else if (current_token == "21") {

        cout << "Error, uso o mala colocacion de '}' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "22") {

        cout << "Error, uso o mala colocacion de '/' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "23") {

        cout << "Error, uso o mala colocacion de '*' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;

```

```

        abort();
    }
    else if (current_token == "24") {

        cout << "Error, uso o mala colocacion de '-' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "25") {

        cout << "Error, uso o mala colocacion de '+' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "26") {

        cout << "Error, uso o mala colocacion de ',' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "27") {

        cout << "Error, uso o mala colocacion de ';' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "28") {

        cout << "Error, uso o mala colocacion de 'ID' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else if (current_token == "29") {

        cout << "Error, uso o mala colocacion de 'Int' en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
    else{
        cout << "Error Match en la linea " << vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}
}

```

```
// <-----Funciones de cada produccion----->
void args_list_prime() {
    if (current_token == "26") {
        match("26");
        arithmetic_expression();
        args_list_prime();
    }
    else if (current_token == "17") {
        return;
    }
    else {
        cout << "Error args_list_prime en la linea " << vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

void args() {
    if (current_token == "16" || split_c_forif(current_token) == "28" || split_c_forif(current_token)
== "29") {
        arithmetic_expression();
        args_list_prime();
    }
    else {
        cout << "Error, argumentos mal formados en la linea " << vecLinea.getIdent(vecParPos +
2) << endl;
        abort();
    }
}

void call_prime() {
    if (current_token == "16" || split_c_forif(current_token) == "28" || split_c_forif(current_token)
== "29" || current_token == "26") {
        args();
        match("17");
    }
    else if (current_token == "17") {
        match("17");
    }
    else {
        cout << "Error, llamada mal formada en la linea " << vecLinea.getIdent(vecParPos + 2) <<
endl;
        abort();
    }
}
```

```

}

void factor_prime() {
    if (current_token == "18") {
        match("18");
        arithmetic_expression();
        match("19");
    }
    else if (current_token == "16") {
        match("16");
        call_prime();
    }
    else if (current_token == "27" || current_token == "17" || current_token == "12" ||
current_token == "10" || current_token == "9" || current_token == "14" ||
        current_token == "13" || current_token == "15" || current_token == "26" || current_token ==
"19" || current_token == "25" || current_token == "24" ||
        current_token == "23" || current_token == "22") {
        return;
    }
    else {
        cout << "Error, ningun operador despues de 'Identificador' mal formado en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

void factor() {
    if (current_token == "16") {
        match("16");
        arithmetic_expression();
        match("17");
    }
    else if (split_c_forif(current_token) == "28") {
        split_c(current_token);
        match("28");
        factor_prime();
    }
    else if (split_c_forif(current_token) == "29") {
        split_c(current_token);
        match("29");
    }
    else {
        cout << "Error, factor mal formado en la linea " << vecLinea.getIdent(vecParPos + 2) <<
endl;
    }
}

```

```

        abort();
    }
}

void mulop() {
    if (current_token == "23") {
        match("23");
    }
    else if (current_token == "22") {
        match("22");
    }
    else {
        cout << "Error, '*' o '/' faltantes en la linea " << vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

void term_prime() {
    if (current_token == "22" || current_token == "23") {
        mulop();
        factor();
        term_prime();
    }
    else if (current_token == "27" || current_token == "17" || current_token == "12" ||
current_token == "10" || current_token == "9" || current_token == "14" ||
current_token == "13" || current_token == "15" || current_token == "26" || current_token ==
"19" || current_token == "25" || current_token == "24") {
        return;
    }
    else {
        cout << "Error, operadores faltantes o termino de expression faltante(s) en la linea " <<
vecLinea.getIdent(vecParPos-1) << endl;
        abort();
    }
}

void term() {
    if (current_token == "16") {
        match("16");
        arithmetic_expression();
        match("17");
        term_prime();
    }
    else if (split_c_forif(current_token) == "28") {

```



```

        split_c(current_token);
        match("28");
        factor_prime();
        term_prime();
    }
    else if (split_c_forif(current_token) == "29") {
        split_c(current_token);
        match("29");
        term_prime();
    }
    else {
        cout << "Error, termino mal formado en la linea " << vecLinea.getIdent(vecParPos + 2) <<
endl;
        abort();
    }
}

```

```

void arithmetic_expression_prime() {
    if (current_token == "25") {
        match("25");
        term();
        arithmetic_expression_prime();
    }
    else if (current_token == "24") {
        match("24");
        term();
        arithmetic_expression_prime();
    }
    else if (current_token == "27" || current_token == "17" || current_token == "12" ||
current_token == "10" ||
        current_token == "9" || current_token == "14" || current_token == "13" || current_token ==
"15" || current_token == "26" || current_token == "19") {
        return;
    }
    else {
        cout << "Error, expresion aritmetica despues de suma o resta mal formada en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

```

```

void arithmetic_expression() {
    if (current_token == "16") {
        match("16");
    }
}

```

```

    arithmetic_expression();
    match("17");
    term_prime();
    arithmetic_expression_prime();
}
else if (split_c_forif(current_token) == "28") {
    // vars
    if (vecOutput.getIdent(vecParPos + 1) == "16") {
        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Llamada a funcion");
        num_llamadas_funciones++;
        current_var++;
        split_c(current_token);
        match("28");
        factor_prime();
        term_prime();
        arithmetic_expression_prime();
    }
    else {
        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Variable Local en
uso");
        num_uso_variable++;
        current_var++;
        split_c(current_token);
        match("28");
        factor_prime();
        term_prime();
        arithmetic_expression_prime();
    }
}
else if (split_c_forif(current_token) == "29") {
    split_c(current_token);
    match("29");
    term_prime();
    arithmetic_expression_prime();
}
else {
    cout << "Error, expresion aritmetica mal formada en la linea " <<
vecLinea.getIdent(vecParPos) << endl;
    abort();
}
}

void expression_prime() {

```

```

if (current_token == "12") {
    match("12");
    arithmetic_expression();
}
else if (current_token == "10") {
    match("10");
    arithmetic_expression();
}
else if (current_token == "9") {
    match("9");
    arithmetic_expression();
}
else if (current_token == "14") {
    match("14");
    arithmetic_expression();
}
else if (current_token == "13") {
    match("13");
    arithmetic_expression();
}
else if (current_token == "15") {
    match("15");
    arithmetic_expression();
}
else if (current_token == "17" || current_token == "27") {
    return;
}
else {
    cout << "Error, mal uso de operadores en expression en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
}

void expression() {
    if (current_token == "16" || split_c_forif(current_token) == "28" || split_c_forif(current_token)
== "29") {
        arithmetic_expression();
        expression_prime();
    }
    else {
        cout << "Error, expresion aritmetica mal formada en la linea " <<
vecLinea.getIdent(vecParPos) << endl;
        abort();
    }
}

```

```

    }
}

void return_stmt_prime() {
    if (current_token == "27") {
        match("27");
    } else if (current_token == "16" || split_c_forif(current_token) == "28" ||
split_c_forif(current_token) == "29") {
        arithmetic_expression();
        expression_prime();
        match("27");
    }
    else {
        cout << "Error, enunciado de return malformado en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

// Else problema
// Se quitó else siendo este el token '4' de la produccion de elseif
void selection_stmt_prime() {
    if (current_token == "4") {
        match("4");
        statement();
    }
    else if (split_c_forif(current_token) == "28" || current_token == "20" || current_token == "21" ||
current_token == "1" || current_token == "2" || current_token == "5" || current_token == "8"
|| current_token == "7") {
        return;
    }
    else {
        cout << "Error, enunciado de seleccion malformado en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

void var_prime() {
    vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Variable Local en uso");
    current_var++;
    num_uso_variable++;
    if (current_token == "18") {
        match("18");
    }
}

```

```

        arithmetic_expression();
        match("19");
    }
    else if (current_token == "27") {
        return;
    }
    else {
        cout << "Error, variable malformada en la linea " << vecLinea.getIdent(vecParPos + 2) <<
endl;
        abort();
    }
}

```

```

void statement_prime() {
    if (current_token == "18") {
        // vars
        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Variable Local en uso");
        num_uso_variable++;
        current_var++;
        match("18");
        arithmetic_expression();
        match("19");
        match("11");
        expression();
        match("27");
    }
    else if (current_token == "11") {
        // vars
        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Variable Local en uso");
        num_uso_variable++;
        current_var++;
        match("11");
        expression();
        match("27");
    }
    else if (current_token == "16") {
        // vars
        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Llamada a funcion");
        num_llamadas_funciones++;
        current_var++;
        match("16");
        call_prime();
        match("27");
    }
}

```

```

    else {
        cout << "Error, variable o funcion definida/declarada incorrectamente en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

void statement() {
    if (split_c_forif(current_token) == "28") {
        split_c(current_token);
        match("28");
        statement_prime();
    }
    else if (current_token == "20") {
        match("20");
        local_declarations();
        statement_list();
        match("21");
    }
    else if (current_token == "5") {
        match("5");
        match("16");
        expression();
        match("17");
        statement();
        selection_stmt_prime();
    }
    else if (current_token == "2") {
        match("2");
        match("16");
        expression();
        match("17");
        statement();
    }
    else if (current_token == "1") {
        match("1");
        return_stmt_prime();
    }
    else if (current_token == "7") {
        match("7");
        split_c(current_token);
        match("28");
        var_prime();
        match("27");
    }
}

```

```

    }
    else if (current_token == "8") {
        match("8");
        expression();
        match("27");
    }
    else {
        cout << "Error, enunciado mal formado en la linea " << vecLinea.getIdent(vecParPos + 2)
<< endl;
        abort();
    }
}

```

```

void statement_list() {
    if (split_c_forif(current_token) == "28") {
        split_c(current_token);
        match("28");
        statement_prime();
        statement_list();
    }
    else if (current_token == "20") {
        match("20");
        local_declarations();
        statement_list();
        match("21");
        statement_list();
    }
    else if (current_token == "5") {
        match("5");
        match("16");
        expression();
        match("17");
        statement();
        selection_stmt_prime();
        statement_list();
    }
    else if (current_token == "2") {
        match("2");
        match("16");
        expression();
        match("17");
        if(current_token != "20"){

```

```

        cout << "Error, abrir llaves en funcion while en la linea "<<
vecLinea.getIdent(vecParPos) << endl;
        abort();
    }
    else {
        statement();
        statement_list();
    }
}
else if (current_token == "1") {
    match("1");
    return_stmt_prime();
    statement_list();
}
else if (current_token == "7") {
    match("7");
    split_c(current_token);
    match("28");
    var_prime();
    match("27");
    statement_list();
}
else if (current_token == "8") {
    match("8");
    expression();
    match("27");
    statement_list();
}
else if (current_token == "21") {
    return;
}
else {
    cout << "Error, Cerrar llaves en la linea " << vecLinea.getIdent(vecParPos) << endl;
    abort();
}
}

void local_declarations() {
    if (current_token == "6") {
        match("6");
        split_c(current_token);
        match("28");
        var_declaration_prime();
        local_declarations();
    }
}

```



```

    }
    else if (current_token == "30") {
        cout << "Error, falta cerrar llaves } " << endl;
        abort();
    }
    else if (split_c_forif(current_token) == "28" || current_token == "20" || current_token == "21" ||
        current_token == "1" || current_token == "2" || current_token == "5" ||
        current_token == "7" || current_token == "8") {
        return;
    }
    else {
        cout << "Error, declaracion de variable o funcion local mal formadas en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

void param_prime() {
    if (current_token == "17" || current_token == "26") {
        return;
    }
    else if (current_token == "18") {
        match("18");
        match("19");
    }
    else {
        cout << "Error, braquetas mal formadas despues de parametros en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

void param() {
    if (current_token == "6") {
        match("6");
        split_c(current_token);
        match("28");

        num_param++;
        // Vars

        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Parametro(s) de la
funcion ");
        current_var++;
    }
}

```

```

        param_prime();
    }
    else {
        cout << "Error, parametro mal formado en la linea " << vecLinea.getIdent(vecParPos + 2)
<< endl;
        abort();
    }
}

```

```

void param_list_prime() {
    if (current_token == "26") {
        match("26");
        param();
        param_list_prime();
    }
    else if (current_token == "17") {
        return;
    }
    else {
        cout << "Error, lista de parametros mal formada en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}

```

```

void params() {
    if (current_token == "6") {
        param();
        param_list_prime();
    }
    else if (current_token == "3") {
        match("3");
    }
    else {
        cout << "Error, parametros mal establecidos en la linea " << vecLinea.getIdent(vecParPos
+ 1) << endl;
        abort();
    }
}

```

```

void var_declaration_prime() {

    if (current_token == "27") {
        // vars

```

```

        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Variable Locales");
        num_var_local++;
        current_var++;
        match("27");
    }
    else if (current_token == "18") {
        // Vars
        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Variable Locales");
        num_var_local++;
        current_var++;
        match("18");
        split_c(current_token);
        match("29");
        match("19");
        match("27");
    }
    else {
        cout << "Error, declaracion de variable mal formada en la linea " <<
vecLinea.getIdent(vecParPos+1) << endl;
        abort();
    }
}

void declaration_prime() {
    if (current_token == "27") {
        // Vars
        match("27");
        int x;
        for (x = 0; x < vecVarGlob.getActualPos(); ) {
            if (vecLetra.getIdent(current_var) == vecVarGlob.getIdent(x)) {
                cout << "No pueden existir mas de una variable global con el mismo nombre , linea "
<< vecLinea.getIdent(vecParPos) << endl;
                abort();
            }
            x++;
        }
        vecVarGlob.insertar(vecVarGlob.getActualPos(), vecLetra.getIdent(current_var));

        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Variable global");
        num_var_global++;
        current_var++;

    }
    else if (current_token == "18") {

```

```

    match("18");
    split_c(current_token);
    match("29");
    match("19");
    match("27");
    // Vars
    int x;
    for (x = 0; x < vecVarGlob.getActualPos();) {
        if (vecLetra.getIdent(current_var) == vecVarGlob.getIdent(x)) {
            cout << "No pueden existir mas de una variable global con el mismo nombre, linea "
<< vecLinea.getIdent(vecParPos) << endl;
            abort();
        }
        x++;
    }
    vecVarGlob.insertar(vecVarGlob.getActualPos(), vecLetra.getIdent(current_var));
    vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Variable global");
    num_var_global++;
    current_var++;

}
else if (current_token == "16") {
    // Vars
    int x;
    for (x = 0; x < vecFunc.getActualPos();) {
        if (vecLetra.getIdent(current_var) == vecFunc.getIdent(x)) {
            cout << "No pueden existir mas de una funcion con el mismo nombre, linea " <<
vecLinea.getIdent(vecParPos) << endl;
            abort();
        }
        x++;
    }
    vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Funcion, INT");
    vecFunc.insertar(vecFunc.getActualPos(), vecLetra.getIdent(current_var));
    num_func++;
    current_var++;
    match("16");
    params();
    match("17");
    if (current_token != "20") {
        cout << "Error, no se abrieron llaves en la funcion en la linea " <<
vecLinea.getIdent(vecParPos) << endl;
        abort();
    }
}

```

```

else {

    vecParams.insertar(vecParams.getActualPos(), " Funcion, INT---> " +
vecFunc.getIdent(vecFunc.getActualPos() - 1) + " ----> Numero de Parametros----> " + "{" +
to_string(num_param) + "}");
    num_param = 0;

    match("20");
    local_declarations();
    statement_list();
    match("21");
}
}
else {
    cout << "Error, declaracion mal formada en la linea " << vecLinea.getIdent(vecParPos + 2)
<< endl;
    abort();
}
}

void declaration_list_prime() {
    if (current_token == "6") {
        match("6");
        split_c(current_token);
        match("28");
        declaration_prime();
        declaration_list_prime();
    }
    else if (current_token == "3") {
        match("3");
        split_c(current_token);
        match("28");
        int x;
        for (x = 0; x < vecFunc.getActualPos(); x) {
            if (vecLetra.getIdent(current_var) == vecFunc.getIdent(x)) {
                cout << "No pueden existir mas de una funcion con el mismo nombre, linea " <<
vecLinea.getIdent(vecParPos) << endl;
                abort();
            }
            x++;
        }
        vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Funcion , VOID");
        vecFunc.insertar(vecFunc.getActualPos(), vecLetra.getIdent(current_var));
        num_func++;
    }
}

```

```

        current_var++;
        match("16");
        params();
        match("17");
        if (current_token != "20") {
            cout << "No se abrieron llaves en funcion en la linea " << vecLinea.getIdent(vecParPos)
<< endl;
            abort();
        }
        else {
            vecParams.insertar(vecParams.getActualPos(), " Funcion , VOID----> " +
vecFunc.getIdent(vecFunc.getActualPos() - 1) + " ----> Numero de Parametros----> " + "{" +
to_string(num_param) + "}");
            num_param = 0;
            match("20");
            local_declarations();
            statement_list();
            match("21");
            declaration_list_prime();
        }

    }
    else if (current_token == "30") {
        if (vecFunc.getIdent(vecFunc.getActualPos()-1) == "Main" &
vecOutput.getIdent(vecOutput.getActualPos() - 2) == "21" ||
vecFunc.getIdent(vecFunc.getActualPos()-1) == "main" &
vecOutput.getIdent(vecOutput.getActualPos() - 2) == "21") {
            return;
        }
        else {
            cout << "Error, favor de terminar con funcion 'main'"<< endl;
            abort();
        }
    }
    else {
        cout << "Error, declaracion de funcion incorrecta en la linea " <<
vecLinea.getIdent(vecParPos) << endl;
        abort();
    }
}

void program_prime() {
    if (current_token == "6") {
        param();
    }
}

```

```

    param_list_prime();
    match("17");
    if (current_token != "20") {
        cout << "Error, abrir llaves en funcion en la linea " << vecLinea.getIdent(vecParPos) <<
endl;
        abort();
    }
    else {
        match("20");
        local_declarations();
        statement_list();
        match("21");
        declaration_list_prime();
    }

}
else if (current_token == "3") {
    match("3");
    match("17");
    match("20");
    local_declarations();
    statement_list();
    match("21");
    declaration_list_prime();
}
else {
    cout << "Error, programa despues de declaracion con 'void' malformado en la linea " <<
vecLinea.getIdent(vecParPos + 2) << endl;
    abort();
}
}

```

```

void program() {
    if (current_token == "6") {
        match("6");
        split_c(current_token);
        match("28");
        declaration_prime();
        declaration_list_prime();
    }
    else if (current_token == "3") {
        match("3");
        split_c(current_token);
        match("28");
    }
}

```

```

int x;
for (x = 0; x < vecFunc.getActualPos();) {
    if (vecLetra.getIdent(current_var) == vecFunc.getIdent(x)) {
        cout << "No pueden existir mas de una funcion con el mismo nombre, linea " <<
vecLinea.getIdent(vecParPos) << endl;
        abort();
    }
    x++;
}
vecVar.insertar(current_var, vecLetra.getIdent(current_var) + " -> Funcion , VOID");
vecFunc.insertar(vecFunc.getActualPos(), vecLetra.getIdent(current_var));
num_func++;
current_var++;
match("16");
program_prime();
}
else {
    if (current_token == "30") {
        cout << "Error, el programa debe ser inicializado con declaraciones 'int' o 'void'" << endl;
        abort();
    }
    else {
        cout << "Error, el programa debe ser inicializado con declaraciones 'int' o 'void' en la
linea " << vecLinea.getIdent(vecParPos + 2) << endl;
        abort();
    }
}
}
//<-----Fin de funciones de cada produccion----->

// Main del Parser
void parserMain() {
    // Asignacion del num 30 de token id como el signo de "$"
    string token_ini = "30";
    vecParPos = 0;
    vecOutput.insertar(vecOutput.getActualPos(), token_ini);
    current_token = vecOutput.getIdent(vecParPos);
    program();

    if (current_token == token_ini) {
        cout << "<-----Mensaje de aceptacion----->" << endl;
        cout << "Syntax_Analysis_OK :D" << endl << endl;
    }
    else {

```



```

        cout << "<-----Mensaje de Error----->" << endl;
        cout << "Syntax_Analysis_Error D:" << endl;
    }
}

// Recorre estados y luego imprime el vectorOutput para obtener los resultados
// Main Principal
int main()
{
    recorrerEstados();
    parserMain();
    //cout << linea_actual << endl;

    cout << "<-----Tablas de simbolos----->" << endl;
    for (int x = 0; x < vecVar.getActualPos();) {
        cout << "< " + vecVar.getIdent(x) + " >" << endl;
        x++;
    }

    cout << endl;
    cout << "<-----Numero de Funciones y Variables----->" << endl;
    cout << endl;
    cout << "  Numero de Funciones ----> " << "{" << num_func << "}" << endl;
    cout << "  Numero de Variables locales ----> " << "{" << num_var_local << "}" << endl;
    cout << "  Numero de Uso de variables locales ----> " << "{" << num_uso_variable << "}" <<
endl;
    cout << "  Numero de Llamadas a funcion ----> " << "{" << num_llamadas_funciones << "}"
<< endl;
    cout << "  Numero de Variables globales ----> " << "{" << num_var_global << "}" <<
endl<<endl;

    cout << "<-----Numero de parametros de cada funcion----->" << endl
<< endl;
    for (int x = 0; x < vecParams.getActualPos();) {
        cout << vecParams.getIdent(x) << endl;
        x++;
    }
}

```

5. Verificación y Validación

5.1. Casos de Prueba

# Caso de Prueba	Descripción	Datos del Test	Resultado Esperado	Resultado Generado	Falló/Pasó
1	No se inserta nada en el input.	Entrada: Vacía	Error, el programa debe ser inicializado con declaraciones 'int' o 'void'	Error, el programa debe ser inicializado con declaraciones 'int' o 'void'	Pasó
2	Recibir el ejemplo completo de input "Sample.txt"	Entrada: Input completo de "Sample.txt"	Mensaje de Sintaxis Correcto, Tabla de símbolos actualizada y Número de variables locales, variables globales, funciones, uso de variables, llamadas a función y funciones con número de parámetros.	Mensaje de Sintaxis Correcto, Tabla de símbolos actualizada y Número de variables locales, variables globales, funciones, uso de variables, llamadas a función y funciones con número de parámetros.	Pasó
3	Declarar una variable global entero solamente.	Entrada: Int x[10];	Mensaje Error, favor de terminar con función 'main'	Mensaje Error, favor de terminar con función 'main'	Pasó
4	Dejar solamente función de main sin nada dentro de ella.	Entrada: void main(void){ }	Mensaje de Sintaxis Correcto, Tabla de símbolos actualizada y Número de variables locales,	Mensaje de Sintaxis Correcto, Tabla de símbolos actualizada y Número de variables locales,	Pasó

			variables globales, funciones, uso de variables, llamadas a función y funciones con número de parámetros.	variables globales, funciones, uso de variables, llamadas a función y funciones con número de parámetros.	
5	Declarar 2 variables globales con el mismo nombre.	Entrada: Int x[10]; Int x;	No pueden existir mas de una variable global con el mismo nombre	No pueden existir mas de una variable global con el mismo nombre	Pasó
6	Declarar 2 funciones con el mismo nombre.	Entrada: Int miniloc{} Int miniloc{}	No pueden existir mas de una funcion con el mismo nombre	No pueden existir mas de una funcion con el mismo nombre	Pasó
7	Agregar una llave abierta al lado de una existente	Entrada: int miniloc(int a[], int low, int high){ '{'	Error, Cerrar llaves en la linea	Error, Cerrar llaves en la linea	Pasó
8	Quitar llave que abre un while.	Entrada: while(i < high)	Error, abrir llaves en funcion while en la linea	Error, abrir llaves en funcion while en la linea	Pasó
9	Dejar un identificador sin declaración int.	Entrada: i;	Error, variable o funcion definida/declarada incorrectamente en la linea	Error, variable o funcion definida/declarada incorrectamente en la linea	Pasó
10	Quitar asignación entre dos identificadores	Entrada: K Low	Error, variable o funcion definida/decl	Error, variable o funcion definida/decl	Pasó

	s.		arada incorrectame nte en la línea	arada incorrectame nte en la línea	
--	----	--	---	---	--

6. Referencias

“What is Table-Driven Design?”. Techopedia.com. 2022.

<https://www.techopedia.com/definition/30408/table-driven-design#:~:text=Table%2DDriven%20Design-,What%20Does%20Table%2DDriven%20Design%20Mean%3F,those%20in%20separate%20external%20tables>. Accesado 29 de April 2022.

“Finite state machines”. isaacomputerscience.org. 2022.

https://isaacomputerscience.org/concepts/dsa_toc_fsm?examBoard=all&stage=all
Accesado 29 de Abril 2022.

R. Castelló, Lectura de Clase, Topic: “Chapter 2 – Lexical Analysis.” TC3048, Escuela de Ingeniería y Ciencias, ITESM, Chihuahua, Chih, Abril, 2022.

SAKSHAM894954, “Tabla de Transición en Autómatas”, acervolima.com. 2022.

<https://es.acervolima.com/tabla-de-transicion-en-automatas/>
Accesado 29 de Abril 2022.

“Diagrama de flujo”, concepto.de. 2022. <https://concepto.de/diagrama-de-flujo/>

Accesado 29 de Abril 2022.

Thomas Hamilton, “How to Write Test Cases: Sample Template with Examples”, guru99.com.

2 de Abril , 2022. <https://www.guru99.com/test-case.html>

Accesado 29 de Abril 2022.

ELLIOT B. KOFFMAN, PAUL A. T. WOLFGANG, “OBJECTS, ABSTRACTION, DATA STRUCTURES AND DESIGN USING C++”, John Wiley & Sons, Inc. ,29 de Abril 2022.