# Procesadores del Lenguaje

Práctica final, entrega final

Autores: Álvaro Marco Pérez (100383382), David Rico Menéndez (100384036)
Datos: Grupo de trabajo 2

# Índice

# 1. Introducción

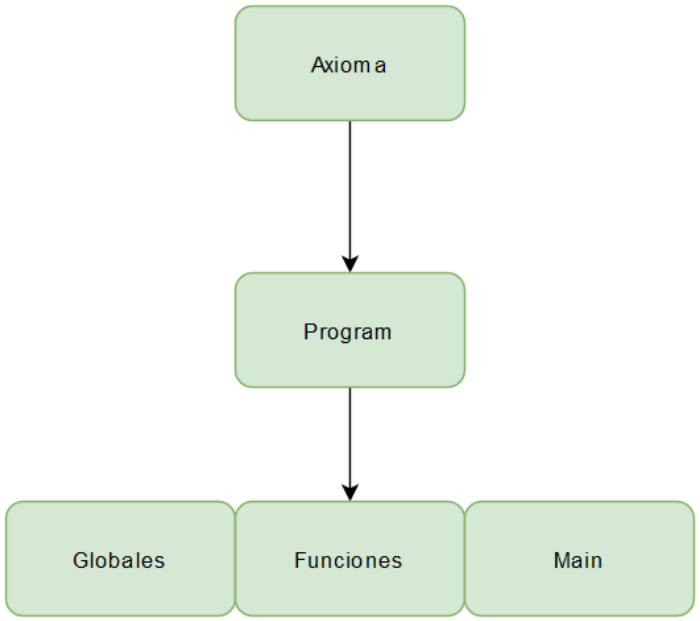
En esta práctica se propone el desarrollo de un traductor de los lenguajes de programación C a LISP. Para el desarrollo de dicho traductor se propone el uso de gramáticas mediante el uso de Bison, una herramienta de creación de compiladores.

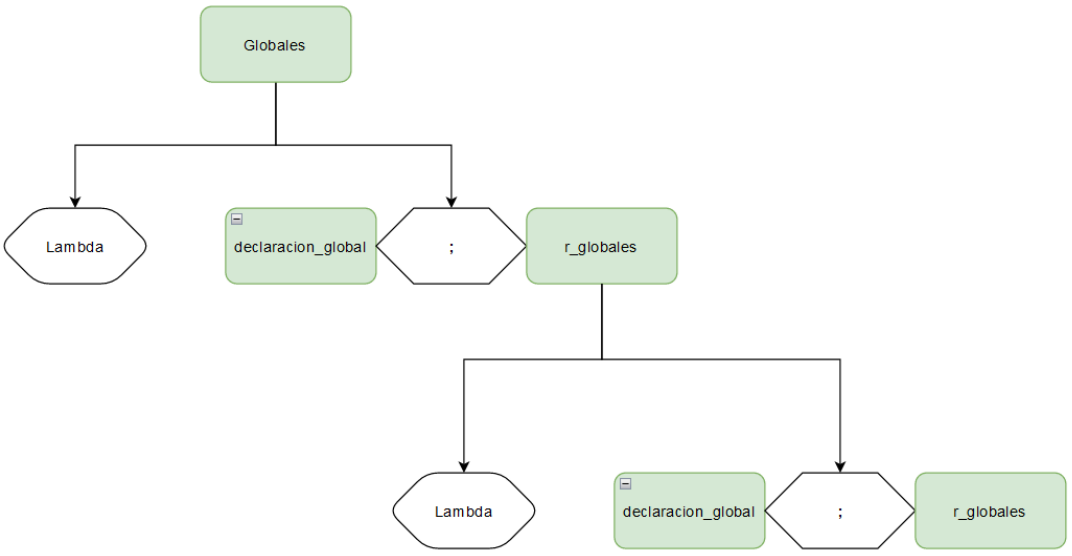
Para la generación de la gramática traductora se contó con un código de base, sobre el que se fueron estructurando las funcionalidades que se requerían semanalmente. A lo largo del desarrollo de la práctica se han ido ampliando y modificando las producciones anteriores con el fin de aumentar el alcance del traductor.

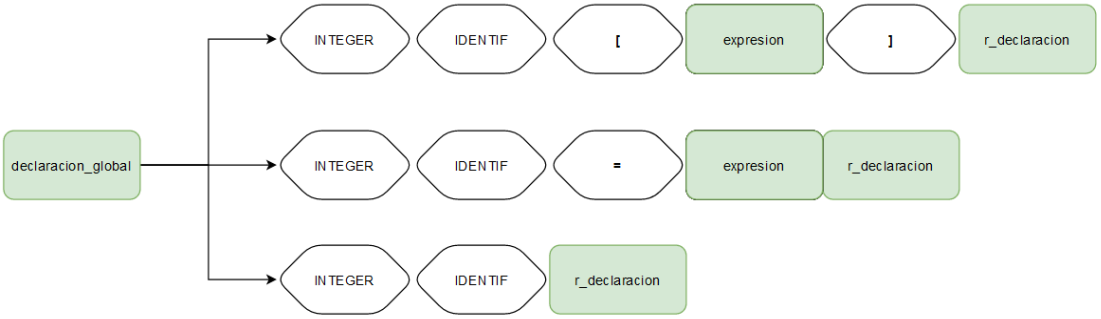
Esta memoria se presentará la gramática final desarrollada, así como su árbol sintáctico y la justificación de las decisiones tomadas por cada apartado. Adicionalmente se incluyen los problemas encontrados y las pruebas adicionales realizadas.

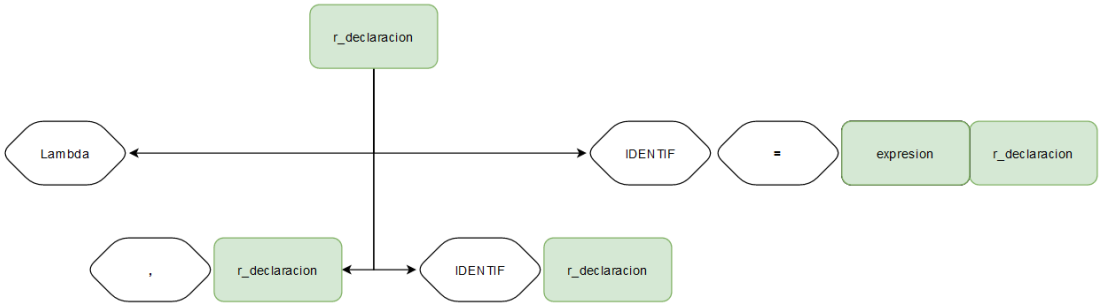
# 2. Árbol sintáctico

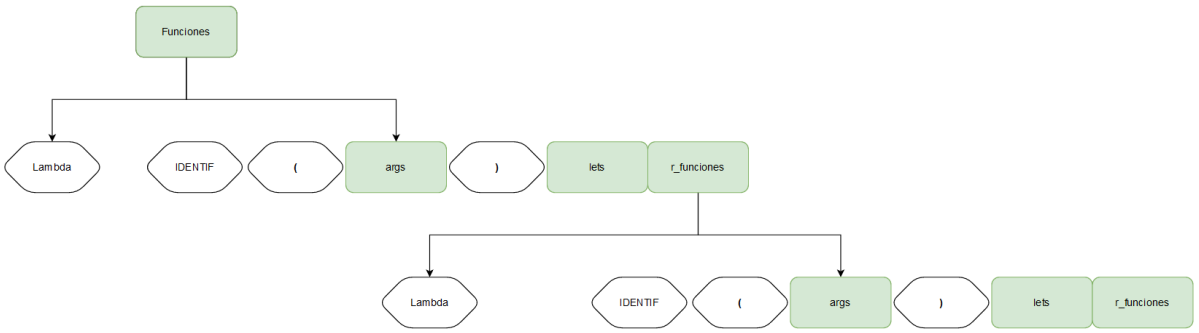
Con el fin de mejorar la comprensibilidad del árbol sintáctico generado por la gramática desarrolla, se descompondrá  en distintas figuras:

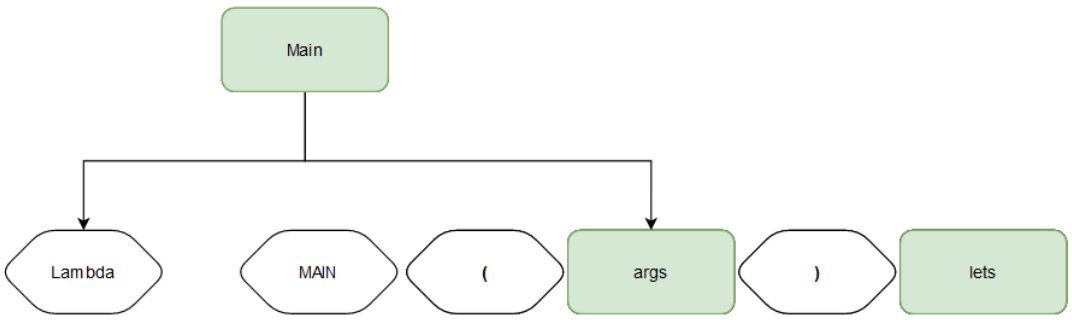


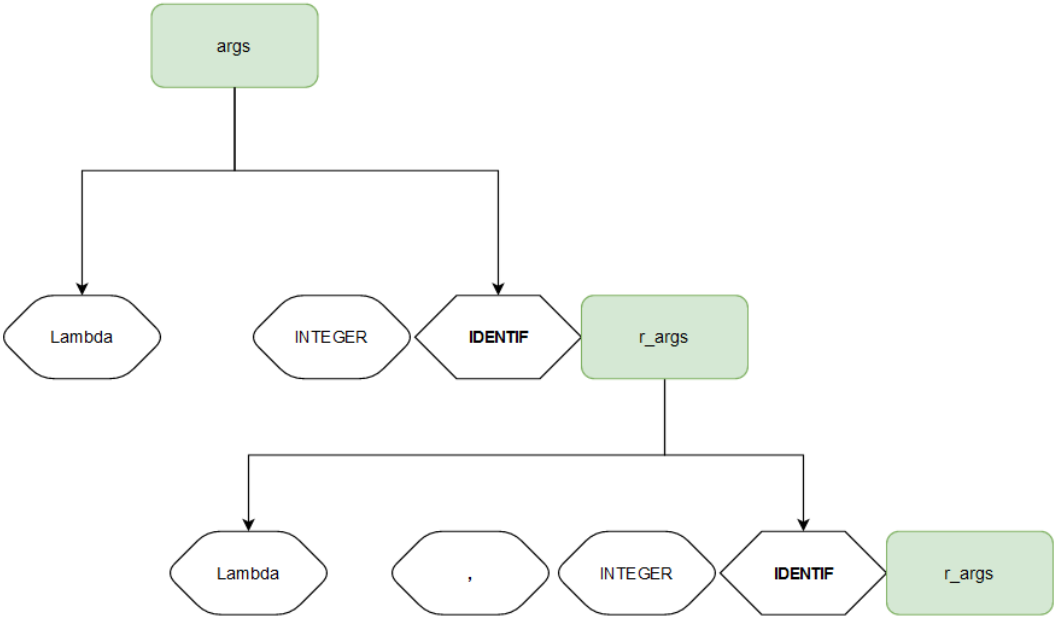



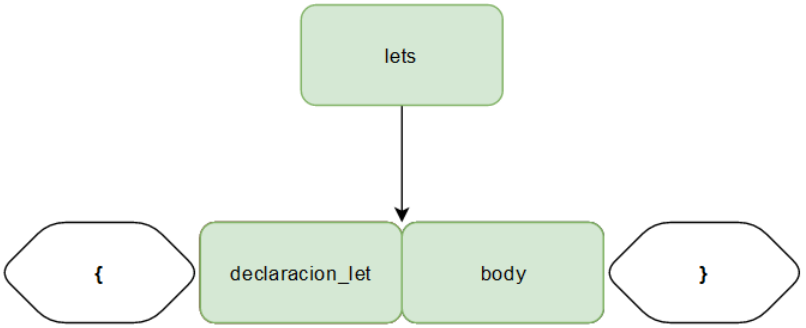

declaracion_global
INTEGER
IDENTIF
[
expresion
]
r_declaracion
INTEGER
IDENTIF
=
expresion
r_declaracion
INTEGER
IDENTIF
r_declaracion

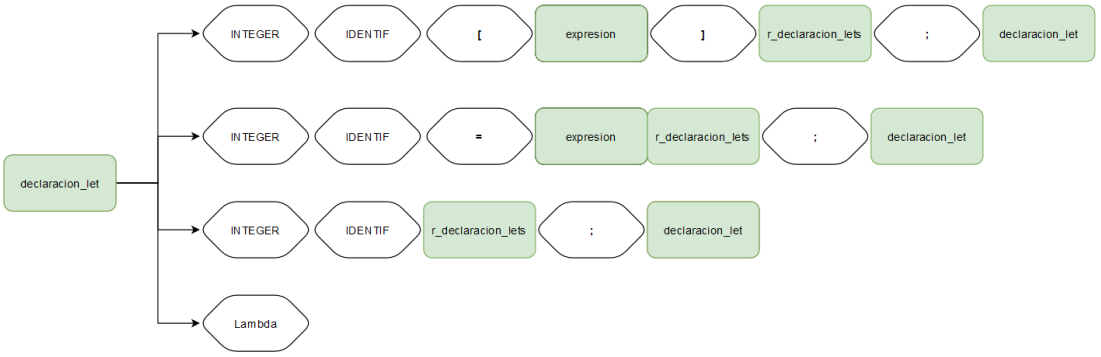

r_declaracion
Lambda
IDENTIF
=
expresion
r_declaracion
,
r_declaracion
IDENTIF
r_declaracion

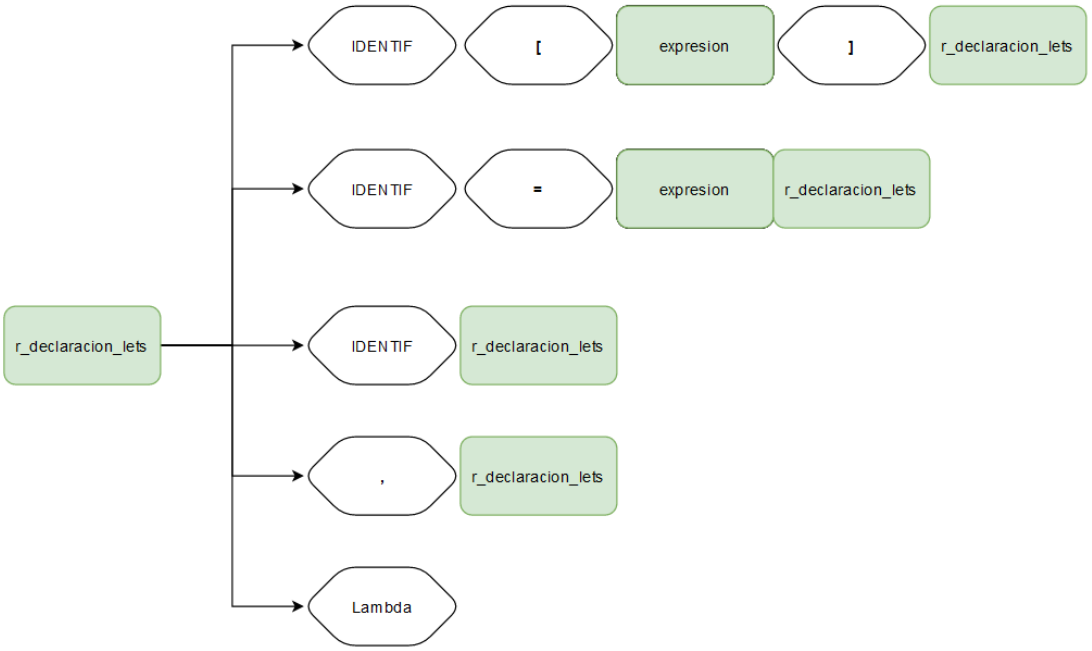

Funciones
Lambda
IDENTIF
(
args
)
lets
r_funciones
Lambda
IDENTIF
(
args
)
lets
r_funciones

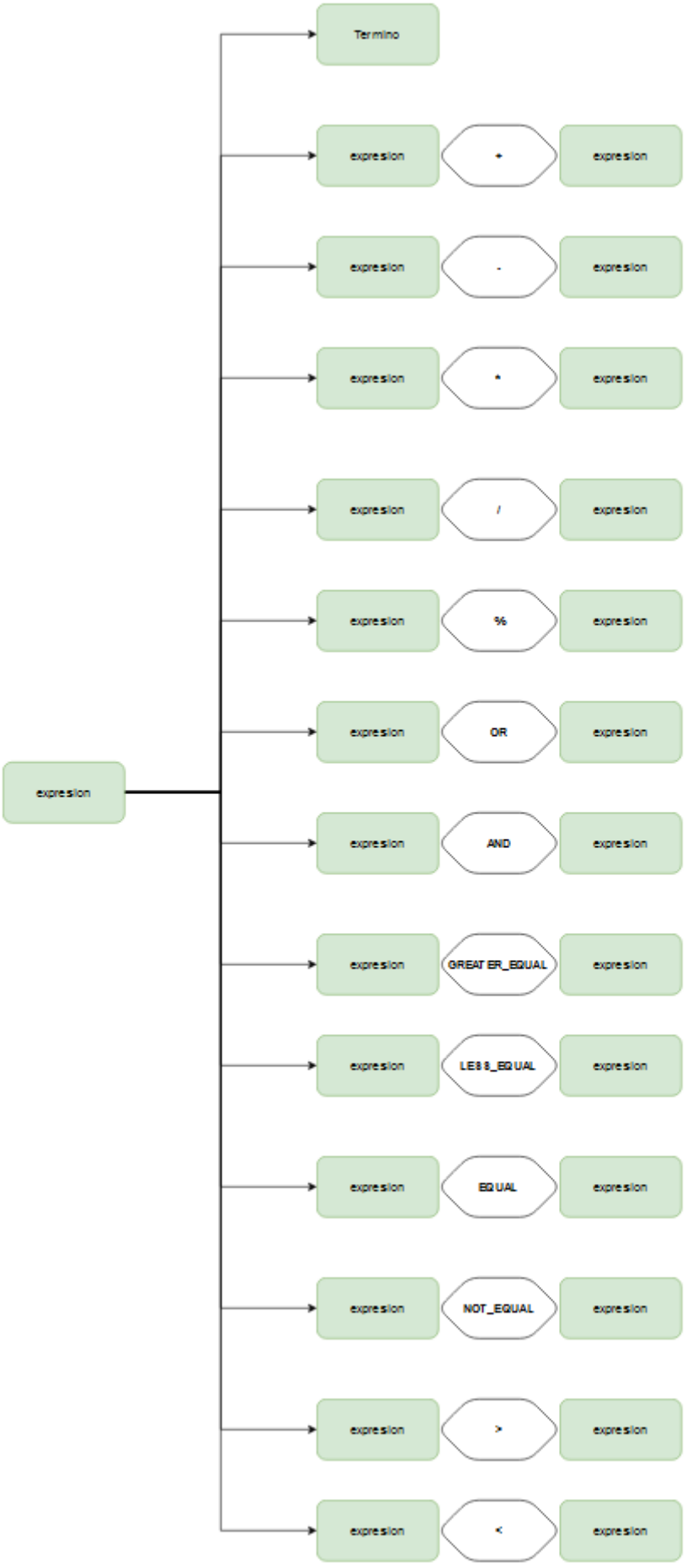

Main
Lambda
MAIN
(
args
)
lets

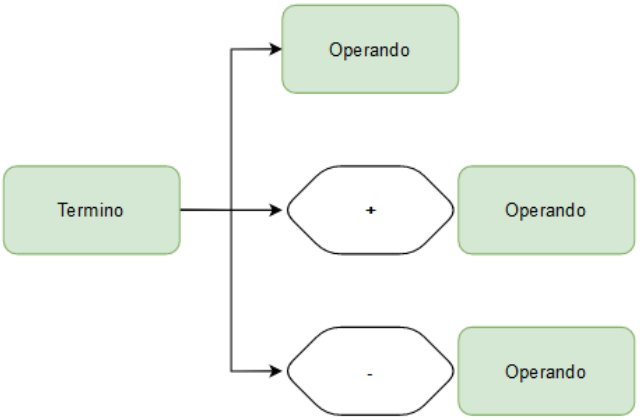

args
Lambda
INTEGER
IDENTIF
r_args
Lambda
,
INTEGER
IDENTIF
r_args

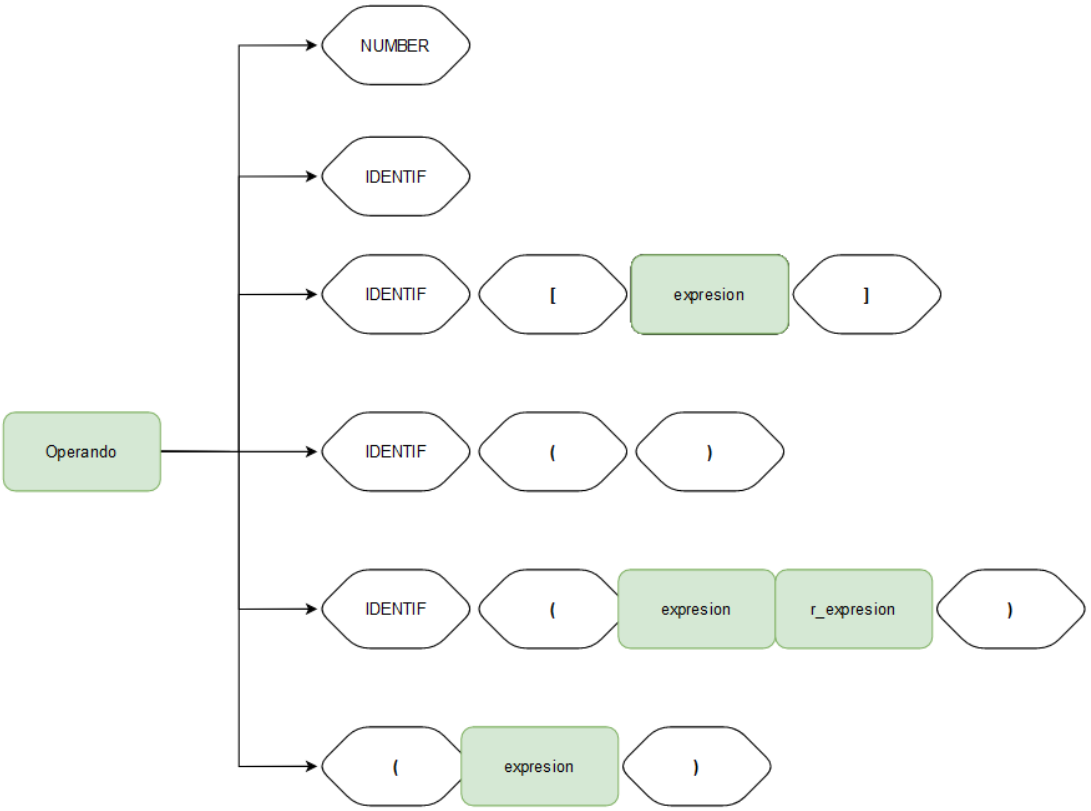

lets
{
declaracion_let
body
}

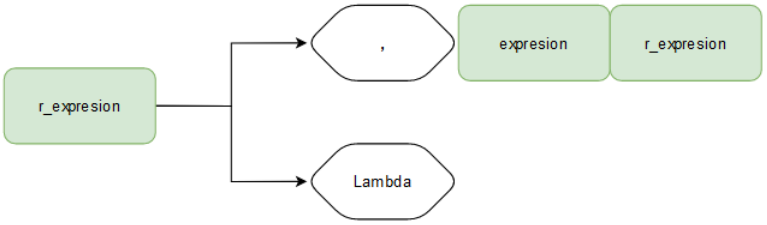

declaracion_let
INTEGER
IDENTIF
[
expresion
]
r_declaracion_lets
;
declaracion_let
INTEGER
IDENTIF
=
expresion
r_declaracion_lets
;
declaracion_let
INTEGER
IDENTIF
r_declaracion_lets
;
declaracion_let
Lambda

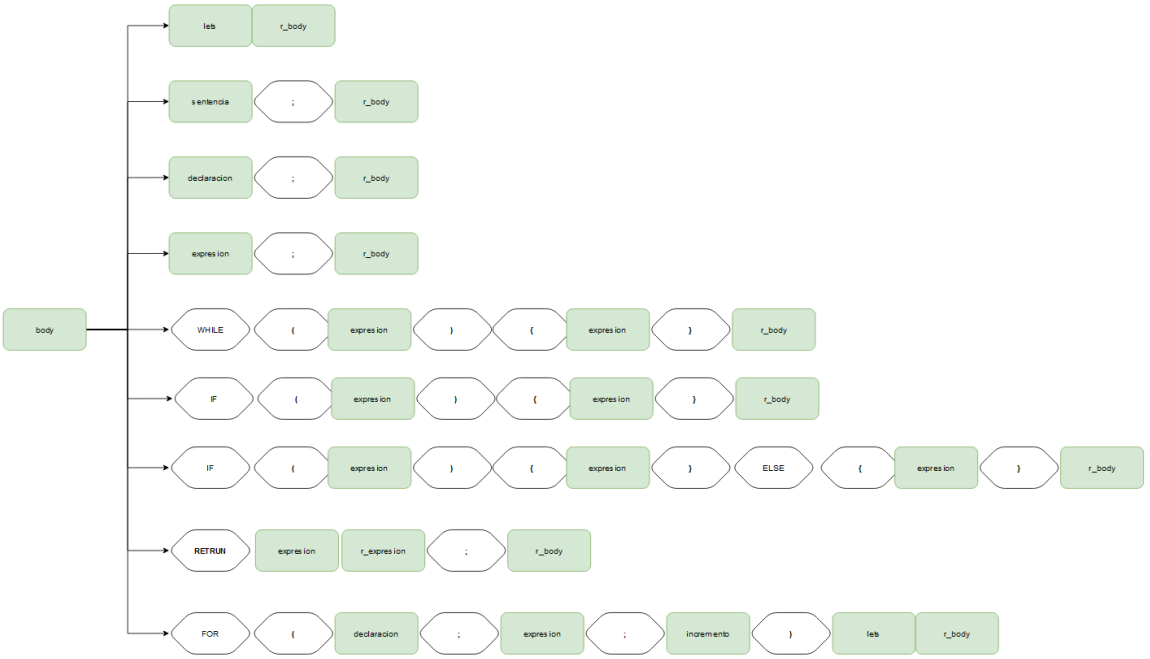

r_declaracion_lets
IDENTIF
[
expresion
]
r_declaracion_lets
IDENTIF
=
expresion
r_declaracion_lets
IDENTIF
r_declaracion_lets
,
r_declaracion_lets
Lambda

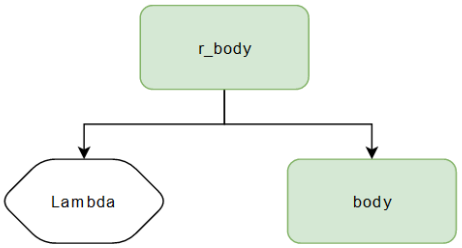

expresion
Termino
expresion
+
expresion
expresion
-
expresion
expresion
*
expresion
expresion
/
expresion
expresion
%
expresion
expresion
OR
expresion
expresion
AND
expresion
expresion
GREATER_EQUAL
expresion
expresion
LESS_EQUAL
expresion
expresion
EQUAL
expresion
expresion
NOT_EQUAL
expresion
expresion
>
expresion
expresion
<
expresion

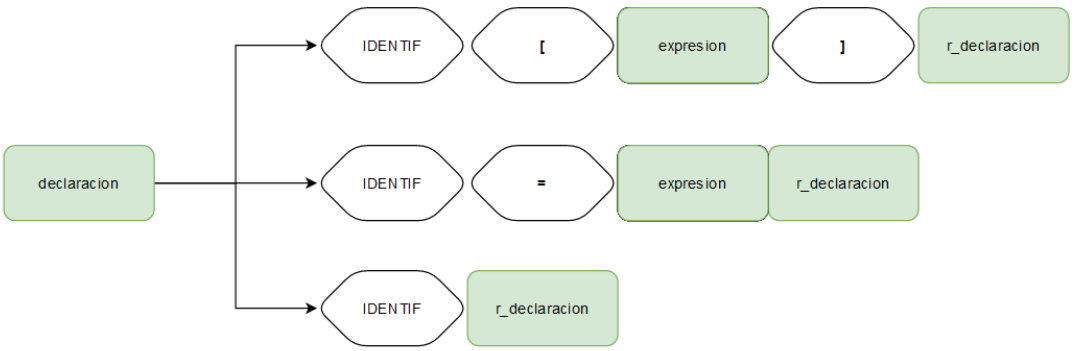

Termino
Operando
+
Operando
-
Operando

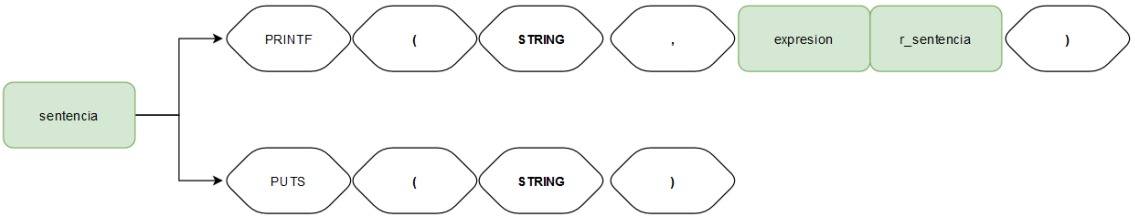

Operando
NUMBER
IDENTIF
IDENTIF
[
expresion
]
IDENTIF
(
)
IDENTIF
(
expresion
r_expresion
)
(
expresion
)

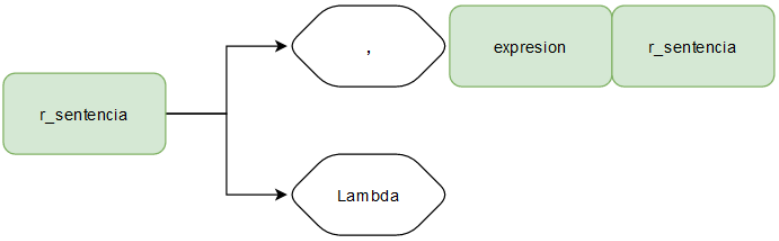

r_expresion
,
expresion
r_expresion
Lambda

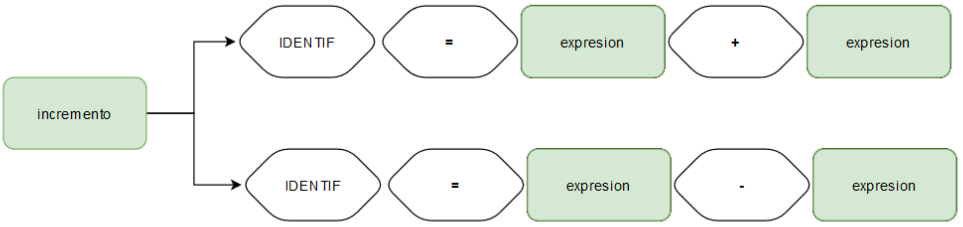

body
lets
r_body
sentencia
;
r_body
declaracion
;
r_body
expresion
;
r_body
WHILE
(
expresion
)
{
expresion
}
r_body
IF
(
expresion
)
{
expresion
}
r_body
IF
(
expresion
)
{
expresion
}
ELSE
{
expresion
}
r_body
RETRUN
expresion
r_expresion
;
r_body
FOR
(
declaracion
;
expresion
;
incremento
)
lets
r_body


r_body
Lambda
body


declaracion
IDENTIF
[
expresion
]
r_declaracion
IDENTIF
=
expresion
r_declaracion
IDENTIF
r_declaracion


sentencia
PRINTF
(
STRING
,
expresion
r_sentencia
)
PUTS
(
STRING
)


r_sentencia
,
expresion
r_sentencia
Lambda


incremento
IDENTIF
=
expresion
+
expresion
IDENTIF
=
expresion
-
expresion

# 3. Alcance de la solución y justificación

La gramática ilustrada en el punto anterior cubre por completo los requerimientos de la práctica.

Sin embargo, para la simplificación del desarrollo de algunos puntos se han tenido en cuenta las siguientes consideraciones:

- Dado que el compilador LISP trabaja indiferentemente con la estructura <return-from ... valor> como con <valor>, se ha decidido implementar solo la primera.
- Dado que el compilador LISP trabaja indiferentemente con la estructura <values valor1 valor2 ...> como con <values valor> y <valor> se ha decidido omitir la última traducción.

Adicionalmente, para la promoción de operaciones aritméticas como booleanas y viceversa, se ha añadido un tipo a la unión proporcionada. Con ella se ha distinguido en los casos necesarios que tipo de operación tenía la producción hija y realizado las transformaciones necesarias.

# 4. Problemas encontrados

En el desarrollo de nuestro traductor para la asignatura de procesadores de lenguaje, nos encontramos con varios problemas que requirieron una atención especial para superarlos.

El primer problema que encontramos fue conseguir un diseño **funcional y jerárquico** que mantuviera una **estructura adecuada**. Específicamente, queríamos que las variables globales fueran definidas antes que las funciones, y que estas fueran definidas antes que la función main. Esto requería una estructura muy clara y bien organizada que, aunque al principio no parecía un problema muy grande, resultó ser más complicado de lo que pensábamos. Finalmente, después de un intenso trabajo de planificación y discusión, logramos llegar a un diseño que cumplía con nuestras expectativas y que, además, era fácil de seguir y entender.

El siguiente problema que encontramos fue durante el desarrollo de las declaraciones múltiples. En particular, queríamos asegurarnos de que nuestra gramática tuviera una **recursividad correcta y adecuada** que nos permitiera manejar declaraciones múltiples sin problemas. Sabíamos que esta era una **parte clave para el posterior desarrollo** y que tendríamos que dedicar tiempo extra para entender cómo hacerlo.

Tras un análisis detallado, logramos encontrar una solución que funcionaba de manera efectiva y que nos permitió avanzar en el desarrollo de nuestro traductor.

Otro punto que nos llevó un tiempo extra, no por su dificultad, sino por el aumento considerable del tamaño de nuestra regla “expresión”, fue la **implementación de todos los operadores**. Sabíamos que era esencial que nuestro traductor pudiera manejar todos los operadores adecuadamente, y que además, **fuera capaz de convertir correctamente entre operadores booleanos y aritméticos** mediante la implementación de un **atributo nuevo llamado "tipo"**. Este atributo nos permitió asegurarnos de que todas las expresiones fueran evaluadas correctamente, y que nuestro traductor fuera capaz de manejar cualquier tipo de operación que se le presentara, incluso dentro de estructuras condicionales.

Finalmente, los últimos dos problemas que encontramos fueron la implementación del "return-from" y la correcta implementación de "let()". En el caso del "return-from", tuvimos que resolver un problema específico que se presentaba al intentar almacenar el nombre de la función en una variable temporal para luego utilizarla en la línea de retorno. Aunque al principio parecía un problema difícil, ya que nuestra propuesta inicial pasaba por realizar un análisis y modificación del string “temp” mediante al uso de un cursor, logramos encontrar una solución que funcionaba de manera adecuada y más sencilla y que nos permitió avanzar en el desarrollo de nuestro traductor. En cuanto a la implementación de "let()", esta fue un punto que nos costó entender al principio debido a las especificaciones solicitadas y su funcionamiento. Sin embargo, tras razonar de manera conjunta las posibilidades y comprobar con el compilador de bison el funcionamiento de esta estructura, logramos entender cómo debía ser implementado y pudimos agregarlo correctamente a nuestro traductor.

# 5. Makefile

El makefile diseñado consta de dos reglas: "make" y "clean". La primera regla, "make", está encargada de compilar el programa y ejecutar las pruebas necesarias. En esta regla se hace uso de las herramientas bison y gcc para compilar el código fuente del programa.
Primero se llama al compilador de bison para generar el analizador sintáctico a partir del archivo "trad.y". La opción "-Wcounterexamples" se utiliza para que bison muestre contraejemplos cuando detecta conflictos en la gramática especificada.
Luego, se llama al compilador de gcc para generar el ejecutable "traductor" a partir del archivo "trad.tab.c" generado por bison.

Finalmente, se ejecutan tres scripts de shell: "borrar_pruebas.sh", "ejecutar_pruebas.sh" y "comparar.sh". A continuación, se explica brevemente que realiza cada uno:

- Borrar_pruebas.sh: Se utiliza para borrar los archivos generados por las pruebas anteriores. Este script está escrito en bash y consta de tres comandos "find" que se utilizan para encontrar y borrar todos los archivos de entrada ".lsp", todos los archivos de salida esperados ".txt" y todos los archivos objeto ".o" en el directorio actual y en sus subdirectorios. Estos comandos se ejecutan después de cambiar el directorio actual a la carpeta "pruebas" donde se encuentran los archivos generados por las pruebas.
- ejecutar _pruebas.sh: Se utiliza para ejecutar las pruebas del programa. Primero, cambia el directorio actual a la carpeta "pruebas". Luego, se ejecutan dos bucles "for" anidados para recorrer los archivos de prueba en las carpetas "iniciales", "avanzadas" y "adicionales". Para cada archivo de prueba ".c", se genera un archivo de salida ".lsp" utilizando el programa de traducción. Si el archivo de salida contiene la cadena "syntax error", se indica que la prueba falló y se incrementa la variable de errores. De lo contrario, se indica que la prueba fue exitosa y se incrementa la variable de éxitos. Al final, se muestra un resumen de las pruebas exitosas y fallidas.
- comparar.sh. El script comienza cambiando al directorio "pruebas". Luego, se ejecutan tres bucles "for" anidados para recorrer los archivos de prueba en las carpetas "iniciales", "avanzadas" y "adicionales". En el primer bucle "for", se compilan los archivos de prueba en C y se ejecutan para generar archivos de salida "_c.txt". En el segundo bucle "for", se ejecutan los archivos de prueba en Lisp para generar archivos de salida "_lisp.txt" correspondientes. En el tercer bucle "for", se comparan los archivos de salida generados por los programas en C y Lisp. Si los archivos coinciden, se indica que la prueba fue exitosa y si no, se indica que la prueba falló. Para esta comparación se decide ignorar espaciados, líneas en blanco y los caracteres de comillas y ‘\n’. Al final, se muestran los resultados de todas las pruebas ejecutadas.

La segunda regla, "clean", se encarga de limpiar los archivos generados por el proceso de compilación. En esta regla se borran el ejecutable "traductor" y el archivo "trad.tab.c" generado por bison. También se ejecuta el script "borrar_pruebas.sh" para borrar los archivos de pruebas.

En resumen, este makefile es muy útil para compilar el código fuente del programa, ejecutar pruebas y limpiar los archivos generados. Además, la opción "-Wcounterexamples" de bison es muy útil para detectar errores en la gramática especificada.

# 6. Pruebas realizadas

## Pruebas dadas por el profesor:

### Básicas:

A continuación, se muestra el output del comparador.

```
Compiling C iniciales/factorial1.c
Executing C iniciales/factorial1.c
Compiling C iniciales/funciones1.c
Executing C iniciales/funciones1.c
Compiling C iniciales/funciones2.c
Executing C iniciales/funciones2.c
Compiling C iniciales/funciones3.c
Executing C iniciales/funciones3.c
Compiling C iniciales/funciones4.c
Executing C iniciales/funciones4.c
Compiling C iniciales/primos1.c
Executing C iniciales/primos1.c
Compiling C iniciales/printf1.c
Executing C iniciales/printf1.c
Compiling C iniciales/printf2.c
Executing C iniciales/printf2.c
Compiling C iniciales/puts1.c
Executing C iniciales/puts1.c
Compiling C iniciales/while1.c
Executing C iniciales/while1.c
Executing LISP iniciales/factorial1.lsp
Executing LISP iniciales/funciones1.lsp
Executing LISP iniciales/funciones2.lsp
Executing LISP iniciales/funciones3.lsp
Executing LISP iniciales/funciones4.lsp
Executing LISP iniciales/primos1.lsp
Executing LISP iniciales/printf1.lsp
Executing LISP iniciales/printf2.lsp
Executing LISP iniciales/puts1.lsp
Executing LISP iniciales/while1.lsp
File iniciales/factorial1_c matches.
File iniciales/funciones1_c matches.
File iniciales/funciones2_c does not match.
File iniciales/funciones3_c matches.
File iniciales/funciones4_c matches.
File iniciales/primos1_c matches.
File iniciales/printf1_c does not match.
File iniciales/printf2_c does not match.
File iniciales/puts1_c matches.
File iniciales/while1_c matches.
```

Como se puede ver, los archivos de texto en funciones 2 y printf1 y printf2 no coinciden, pero esto es algo que esta contemplado, ya que dichas pruebas realizan printf cuyo texto se espera que no se imprima, por lo tanto, **TODAS las pruebas iniciales se consideran SUPERADAS CORRECTAMENTE.**

## Avanzadas:

A continuación, se muestra el output del comparador.

```
Compiling C avanzadas/collatz1.c
Executing C avanzadas/collatz1.c
Compiling C avanzadas/factorial2.c
Executing C avanzadas/factorial2.c
Compiling C avanzadas/fibonacci1.c
Executing C avanzadas/fibonacci1.c
Compiling C avanzadas/fibonacci2.c
Executing C avanzadas/fibonacci2.c
Compiling C avanzadas/fibonacci3.c
Executing C avanzadas/fibonacci3.c
Compiling C avanzadas/funciones5.c
Executing C avanzadas/funciones5.c
Compiling C avanzadas/funciones6.c
Executing C avanzadas/funciones6.c
Compiling C avanzadas/potencias1.c
Executing C avanzadas/potencias1.c
Compiling C avanzadas/primos2.c
Executing C avanzadas/primos2.c
Compiling C avanzadas/primos3.c
Executing C avanzadas/primos3.c
Compiling C avanzadas/primos4.c
Executing C avanzadas/primos4.c
Compiling C avanzadas/primos5.c
Executing C avanzadas/primos5.c
Executing LISP avanzadas/collatz1.lsp
Executing LISP avanzadas/factorial2.lsp
Executing LISP avanzadas/fibonacci1.lsp
Executing LISP avanzadas/fibonacci2.lsp
Executing LISP avanzadas/fibonacci3.lsp
Executing LISP avanzadas/funciones5.lsp
Executing LISP avanzadas/funciones6.lsp
Executing LISP avanzadas/potencias1.lsp
Executing LISP avanzadas/primos2.lsp
Executing LISP avanzadas/primos3.lsp
Executing LISP avanzadas/primos4.lsp
Executing LISP avanzadas/primos5.lsp
File avanzadas/collatz1_c matches.
File avanzadas/factorial2_c matches.
File avanzadas/fibonacci1_c matches.
File avanzadas/fibonacci2_c matches.
File avanzadas/fibonacci3_c matches.
File avanzadas/funciones5_c does not match.
File avanzadas/funciones6_c matches.
File avanzadas/potencias1_c matches.
File avanzadas/primos2_c matches.
File avanzadas/primos3_c matches.
File avanzadas/primos4_c matches.
File avanzadas/primos5_c matches.
```

De nuevo, en este apartado, supera todas salvo la comprobación de funciones5, pero como en el apartado anterior comentamos, se debe al printf y su orden de ejecución, por lo cual, se considera también que **ha superado TODAS las pruebas.**

# Pruebas desarrolladas por los alumnos:

## Adicionales:

Una última vez, se muestra el output del comparador:

```
Compiling C adicionales/1al10.c
Executing C adicionales/1al10.c
Compiling C adicionales/arreglo.c
Executing C adicionales/arreglo.c
Compiling C adicionales/fibo.c
Executing C adicionales/fibo.c
Compiling C adicionales/maximo.c
Executing C adicionales/maximo.c
Compiling C adicionales/suma.c
Executing C adicionales/suma.c
Compiling C adicionales/suma10.c
Executing C adicionales/suma10.c
Executing LISP adicionales/1al10.lsp
Executing LISP adicionales/arreglo.lsp
Executing LISP adicionales/fibo.lsp
Executing LISP adicionales/maximo.lsp
Executing LISP adicionales/suma.lsp
Executing LISP adicionales/suma10.lsp
File adicionales/1al10_c matches.
File adicionales/arreglo_c matches.
File adicionales/fibo_c matches.
File adicionales/maximo_c matches.
File adicionales/suma10_c matches.
File adicionales/suma_c matches.
```

Hemos tenido cuidado de no incluir printfs problemáticos por claridad, y en efecto supera todas las pruebas adicionales propuestas. A continuación contamos brevemente el objetivo de cada una de estas:

- **1al10.c:** Este script comprueba el funcionamiento del for y el prinf de números, devolviendo una cuenta de 1 a 10 por pantalla.
- **arreglo.c:** Este script es de los más complejos propuestos, sirve para comprobar el correcto funcionamiento de los arrays, los bucles anidados y las variables locales, recibiendo como entrad a un array de números desordenados y ordenándolos para finalmente imprimirlos en el orden correcto.
- **fibo.c:** es un programa en lenguaje C que imprime la secuencia de Fibonacci hasta el número 100. La secuencia de Fibonacci comienza con los números 0 y 1, y los siguientes números se generan sumando los dos números anteriores. La dificultad principal es la correcta ejecución de un bucle tan largo, y la igualación de variables final.
- **maximo.c:** Este programa devuelve el máximo entre tres números. El objetivo principal es comprobar la correcta utilización de secuencias if e if-else encadenadas, así como comparadores complejos mezclados con operaciones aritméticas.

- **suma10.c y suma.c:** Son nuestras pruebas más básicas, una suma simple y una suma de todos los números del uno al 10. Se utilizan para comprobar el funcionamiento de los bucles y las reglas de expresión, así como para comprobar el correcto uso de let().

# 7. Conclusiones

El desarrollo de nuestro traductor para la asignatura de Procesadores de Lenguaje ha sido un éxito, superando las pruebas propuestas y las desarrolladas. Hemos tenido la posibilidad de familiarizarnos con bison y reencontrarnos con el desarrollo en C, siendo esto primero más difícil para nosotros, ya que debido a su estructuración y uso del sistema de tokens, hemos tenido que ser muy metódicos y limpios en su desarrollo para facilitar el posterior entendimiento del código.

Este proyecto ha sido una excelente oportunidad para poner en práctica los conocimientos aprendidos en clase, superar los desafíos encontrados y aprender aún más sobre la asignatura.

Aunque al principio fue difícil llegar a un diseño de gramática adecuado, finalmente se logró construir un traductor robusto y confiable. Las decisiones de diseño tomadas fueron cuidadosamente razonadas y explicadas en este report, y consideramos que cumplen satisfactoriamente con los requisitos exigidos en los distintos puntos del proyecto.

# 8. Anexos

## a. Archivo svg Árbol Sintáctico

https://github.com/Davidryt/PL/blob/3f53c98c517da309674bf3943a2aaa80c65591b8/Practica%20Final/trad/Arbol%20sint%C3%A1ctico.drawio.svg

## b. Gramática final

```
1.  axioma:       program

2.  program:      globales funciones main

3.  globales:     /* Lambda */
4.          |     declaracion_global ';' r_globales

5.  r_globales:   /* Lambda */
6.          |     declaracion_global ';' r_globales

7.  funciones:    /* Lambda */
8.          |     IDENTIF  '(' args ')' lets   r_funciones

9.  r_funciones:  /* Lambda */
10.         |     IDENTIF  '(' args ')' lets   r_funciones

11. main:         MAIN '(' args ')' lets

12. args:         /* Lambda */
13.         |     INTEGER IDENTIF  r_args

14. r_args:       /* Lambda */
15.         |     ',' INTEGER IDENTIF r_args

16. body:         sentencia ';' r_body
17.         |     declaracion ';' r_body
18.         |     WHILE '(' expresion ')' '{' body '}' r_body
19.         |     IF '(' expresion ')' '{' body '}' r_body
20.         |     IF '(' expresion ')' '{' body '}' ELSE '{' body '}' r_body
21.         |     FOR '(' declaracion ';' expresion ';' incremento ')' lets r_body
```

```
22.             |       RETURN expresion r_expresion ';' r_body
23.             |       expresion ';' r_body
24.             |       lets  r_body

25.     lets:           '{' declaracion_let  body '}'

26.     declaracion_let:
27.                     /* Lambda */
28.             |       INTEGER IDENTIF '=' expresion r_declaracion_lets ';' declaracion_let
29.             |       INTEGER IDENTIF r_declaracion_lets  ';'  declaracion_let
30.             |       INTEGER IDENTIF '[' expresion ']' r_declaracion_lets ';'
                                declaracion_let

31.     r_declaracion_lets:
                         /* Lambda */
32.             |       ',' r_declaracion_lets
33.             |       IDENTIF '=' expresion r_declaracion_lets
34.             |       IDENTIF r_declaracion_lets
35.             |       IDENTIF '[' expresion ']'  r_declaracion_lets

36.     r_body:         /* Lambda */
37.             |       body

38.     sentencia:      PRINTF '(' STRING ',' expresion r_sentencia ')'
39.             |       PUTS '(' STRING ')'

40.     r_sentencia:    /* Lambda */
41.             |        ',' expresion r_sentencia

42.     declaracion:    IDENTIF '=' expresion
43.             |       IDENTIF '[' expresion ']' '=' expresion r_declaracion
44.             |       IDENTIF ',' r_identif '=' expresion r_expresion

45.     r_identif:      IDENTIF
46.             |       IDENTIF ',' r_identif

47.     declaracion_global:
48.                     INTEGER IDENTIF '=' expresion r_declaracion
49.             |       INTEGER IDENTIF r_declaracion
50.             |       INTEGER IDENTIF '[' expresion ']'  r_declaracion
```

```
51.  r_declaracion:
                    /* Lambda */
52.         |       ',' r_declaracion
53.         |       IDENTIF '=' expresion r_declaracion
54.         |       IDENTIF r_declaracion
55.         |       IDENTIF '[' expresion ']'  r_declaracion
56.         |       IDENTIF '[' expresion ']' '=' expresion r_declaracion

57.  expresion:     termino
58.         |       expresion '+' expresion
59.         |       expresion '-' expresion
60.         |       expresion '*' expresion
61.         |       expresion '/' expresion
62.         |       expresion '%' expresion
63.         |       expresion OR expresion
64.         |       expresion AND expresion
65.         |       expresion GREATER_EQUAL expresion
66.         |       expresion LESS_EQUAL expresion
67.         |       expresion EQUAL expresion
68.         |       expresion NOT_EQUAL expresion
69.         |       expresion '<' expresion
70.         |       expresion '>' expresion

71.  r_expresion:   /* Lambda */
72.         |       ',' expresion r_expresion

73.  incremento:   IDENTIF '='   expresion '+' expresion
74.         |       IDENTIF '='   expresion '-' expresion

75.  termino:       operando
76.         |       '+' operando %prec UNARY_SIGN
77.         |        '-' operando %prec UNARY_SIGN

78.  operando:     IDENTIF
79.         |       NUMBER
80.         |       IDENTIF '[' expresion ']'
81.         |       IDENTIF '('  ')'
82.         |       IDENTIF '(' expresion r_expresion ')'
83.         |       '(' expresion ')'
```