Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

Léxico y Sintáxis

Herramienta Seleccionada

Para poder realizar los Lexers y Parsers, use la herramienta ANTLR ya que nativamente trabaja con Java y al ser mi lenguaje de programación seleccionado es una gran opción, además que es extremadamente sencilla de usar y recibe gramáticas poco simplificadas. Para poder crear el proyecto en Java use Maven ya que permite una conexión con ANTLR a través de plugins, el programa se llama y se encuentra dentro de la carpeta *duck*. Las reglas léxicas y gramaticales están definidas en el formato de ANTLR (.g4) como podemos ver son prácticamente idénticas al formato anterior por lo que crear este archivo fue muy sencillo.

Reglas Léxicas Usadas

- PROGRAM : 'program';
- MAIN : 'main';
- END : 'end';
- VAR : 'var';
- VOID : 'void';
- PRINT : 'print';
- WHILE : 'while';
- DO : 'do';
- IF : 'if';
- ELSE : 'else';
- INT_T : 'int';
- FLOAT_T : 'float';
- ASSIGN : '=';
- GT : '>';
- LT : '<';
- NE : '!=';
- SEMICOLON : ';';
- COMMA : ',';
- COLON : ':';
- LPAREN : '(';
- RPAREN : ')';
- LBRACE : '{';
- RBRACE : '}';
- PLUS : '+';

- MINUS : '-';
- MULT : '*';
- DIV : '/';
- ID : [a-zA-Z_][a-zA-Z_0-9]*;
- CTE_INT : [0-9]+;
- CTE_FLOAT : [0-9]+ '.' [0-9]+;
- CTE_STRING: "" (~["\\] | '\\' .)* "";
- WS : [\t\r\n]+ -> skip;
- LBRACKET: '[';
- RBRACKET: ']';

Agregué el token WS para ignorar espacios, saltos de línea y de párrafos, esto como una medida inicial para afrontar este problema. Si en un futuro decido preocuparme por la indentación puedo agregar otras reglas que involucren estos caracteres.

Gramática Formal

programa : PROGRAM ID SEMICOLON vars funcs MAIN body END ;

vars : VAR varDef | ;
varDef : ID varDefID COLON type SEMICOLON multipleVar ;
varDefID : COMMA ID varDefID | ;
multipleVar : varDef | ;

funcs : VOID ID LPAREN funcArguments RPAREN LBRACKET vars body RBRACKET SEMICOLON funcs | ;
funcArguments : ID COLON type multipleFuncArguments | ;
multipleFuncArguments : COMMA funcArguments | ;

body : LBRACE bodyStatement RBRACE ;
bodyStatement : statement bodyStatement | ;

statement : assign
| condition
| cycle
| f_call
| print ;

assign : ID ASSIGN expresion SEMICOLON ;

Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

```
condition : IF LPAREN expresion RPAREN body elsePart SEMICOLON ;
elsePart : ELSE body | ;

cycle : WHILE LPAREN expresion RPAREN DO body SEMICOLON ;

print : PRINT LPAREN printExpresion multiplePrint RPAREN SEMICOLON ;
printExpresion : expresion | CTE_STRING ;
multiplePrint : COMMA printExpresion multiplePrint | ;

expresion : exp expresionFinal ;
expresionFinal : GT exp
               | LT exp
               | NE exp
               | ;

exp : termino expFinal ;
expFinal : PLUS exp
         | MINUS exp
         | ;

termino : factor terminoFinal ;
terminoFinal : MULT termino
         | DIV termino
         | ;

factor : LPAREN expresion RPAREN
         | sumaResta id_cte ;

sumaResta : PLUS
         | MINUS
         | ;

id_cte : ID
         | CTE_INT
         | CTE_FLOAT ;

f_call : ID LPAREN f_call_expresion RPAREN SEMICOLON ;
f_call_expresion : expresion m_f_call_expresion
```

```
                | ;
m_f_call_expresion : COMMA expresion m_f_call_expresion
                | ;

type : INT_T | FLOAT_T ;
```

Test

Para correr los casos creé un pequeño programa en Java (Main.Java), este permite leer varios archivos .txt con el lenguaje creado (se encuentran en la carpeta *main/resources/casos)* e imprime su estado de aceptación, en el caso de ser rechazados indica la posición y el error encontrado.

Comando para correr programa

```
mvn clean compile exec:java -Dexec.mainClass="com.ok.Main"
```

En el caso de los casos que adjunte los primeros 4 deben de ser aceptados y los 2 últimos rechazados.

Caso 1:

```
program p1;
var x, y : int;
z : float;
main {
  x = 5;
  print(x);
}
end
```

Caso 2:

Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

```
program p2;
void suma(a : int, b : float)[
  var c : float;
  {c = a + b;
  print(c);}
];
main {
  x = 1;
  while (x < 5) do {
    if (x > 0) {
      print("positivo");
    };
    x = x + 1;
  };
}
end
```

Caso 3:

```
program p3;
void saludar(nombre : int)[
  {print("Hola");}
];
main {
  saludar(3);
}
end
```

Caso 4:

Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

```
program p4;
main {
  a = 3.5;
  b = a * (2 + 1.5);
  if (b != 0.0) {
    print("ok");
  };
}
end
```

Caso 5:

```
program p5;
main {
  x = 10
  print(x);
}
end
```

Caso 6:

```
program p6;
main {
  y = 2;
  if y > 1 {
    print("fail");
  };
}
end
```

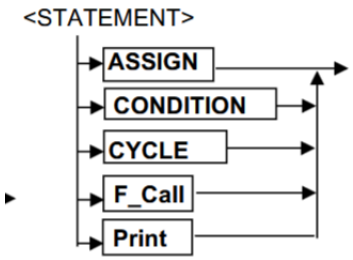
Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

```
Analizando: test1.txt
ACEPTADO

Analizando: test2.txt
ACEPTADO

Analizando: test3.txt
ACEPTADO

Analizando: test4.txt
ACEPTADO

Analizando: test5.txt
RECHAZADO: Error de sintaxis en línea 4:2 -> no viable alternative at input 'print'

Analizando: test6.txt
RECHAZADO: Error de sintaxis en línea 4:5 -> missing '(' at 'y'
```

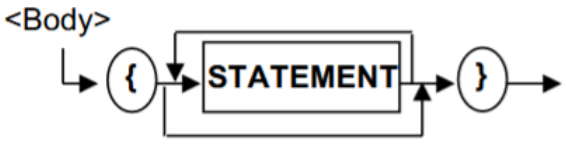
Comentarios

Encontré dos características del lenguaje que no me parecen adecuadas.

- No se pueden poner variables en el BODY:
  Se debe a que STATEMENT no puede adquirir el valor de una variable.



  Esto resulta en que no se pueden definir variables en el main.
- Se tienen que poner corchetes en el BODY de las funciones:
  Debido a la definición del BODY



  Las funciones adquieren esta forma:

Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

```
void suma(a : int, b : float)[
  var c : float;
  {c = a + b;
  print(c);}
];
```

Los corchetes están de más y no son necesarios para poder definir la función.

Estos son aspectos opcionales que podrían ser modificados en entregas posteriores en caso de ser necesario.

Semántica

Estructura Cubo Semántico

Para el cubo semántico use un HashMap de 3 dimensiones que me permite una búsqueda eficiente para validar operaciones entre los dos diferentes tipos de variables que existen en el lenguaje (float e int). En el caso de que sea posible da como resultado el tipo de variable de hacer la operación.

Las operaciones que implementé en este avance: "+", "-", "*", "/", "<", ">", "!=", "=".
Los tipos de variables implementadas: “int” y “float”.

La función “resultado” es la utilizada para hacer llamada a esta estructura en cualquier parte del código y será usada por la semántica en futuros avances.

Estructura Directorio de Funciones

El Directorio de Funciones (Clase) es un HashMap de Funciones (Clase) que a su vez es un HashMap de Variables (Clase). Use HashMaps ya que me permite la búsqueda rápida y eficiente de variables y funciones.

Directorio de Funciones:
Propiedades:

- funciones: HashMap de funciones.

Funciones:

- agregarFuncion(): Validar que la función no exista y agregarla al directorio.
- obtenerFuncion(): Obtener la función a partir del id.
- existeFuncion(): Comprobar que una función no exista a partir del id.

- todas(): Regresa todas las funciones en el directorio.
- toString(): Regresa las funciones como un texto para poder imprimir y hacer pruebas.

Función:
Propiedades:
- tablaVariables: HashMap de variables, se inicializa un HashMap vacío al crear una función.
- parametros: Una lista de los parámetros de la función como id.

Funciones:
- toString(): Regresa la función como un texto para poder imprimir y hacer pruebas.
- agregarVariable(): Validar que la variable no exista y agregala a la tabla.

Variable:
Propiedades:
- tipo: el tipo de la variable (en este caso int o float).

Funciones:
- toString(): Regresa la variable como un texto para poder imprimir y hacer pruebas.

Puntos Neurálgicos

ANTLR genera un árbol de análisis sintáctico a partir del código al ejecutar el parser. Este árbol contiene un nodo por cada regla gramatical reconocida, lo que permite acceder directamente a los elementos (tokens y subreglas) mediante objetos de tipo context.

Debido a esta particularidad de ANTLR ya no es necesario insertar puntos neurálgicos manuales en la gramática para recolectar información semántica, se puede usar un listener generado por ANTLR, el cual reacciona automáticamente al entrar o salir de cada regla.

En este proyecto, se creó un listener personalizado que analiza la semántica del programa durante el recorrido del árbol. Estas son las funciones clave implementadas en el listener:

- enterPrograma: Crea la función principal del programa (guardada con el nombre "global" internamente) y se registra el nombre real del programa para evitar colisiones.

- enterVarDef: Se agrega una variable a la función actual. Si ya existe marca error. Aquí también se comprueba si ya existe una función global con el mismo nombre.
- enterFuncs: Agrega funciones al directorio de funciones, si ya existe marca error. Aquí tambien comprueba que no se llame de la misma forma que el programa.
- procesarArgumentos: Agrega argumentos como variables a la función actual.
- getDirectorio: Esta función me permite obtener el directorio en cualquier punto de la ejecución.
- exitFuncs: Permite salir de la función cuando ya se guardaron todas las variables.

También se implementó en el main un print a partir de la función del Directorio de Funciones toString(), donde imprime el directorio para comprobar su funcionalidad.

```
? Analizando: test3.txt
ACEPTADO
Directorio de Funciones:
DirectorioFunciones{
{saludar=Funcion{tipoRetorno='void', parametros=[int], tablaVariables={nombre=Variable{tipo='int'}}}, global=Funcion{tip
oRetorno='void', parametros=[], tablaVariables={}}, hola=Funcion{tipoRetorno='void', parametros=[int], tablaVariables={n
ombre=Variable{tipo='int'}}}}
}
```

Y en el caso de cometer un error semántico imprimir el problema.

```
? Analizando: test1.txt
RECHAZADO: Variable 'x' ya fue declarada globalmente. No puede redeclararse en 'suma'
```

Comando para correr codigo:

```
mvn exec:java -Dexec.mainClass="com.ok.Main"
```

Generación de Cuadruplos Lineales

Estructuras

Lista de Cuádruplos

Esta es una lista de la clase Cuádruplo, la cual define la estructura intermedia del compilador, está compuesta por un operador(String) que define qué operación se va a realizar, un operando 1(String) y operando 2(String) que son los objetos a los que se les va a aplicar la operación y finalmente un resultado(String) que es el lugar donde se va a almacenar esa operación. Por ejemplo para $x = a + b$, el cuádruplo resultante sería

Cuadruplo("+", a, b, temp1)

Cuadruplo("=", temp1, -1, x)

Stack de operadores (String)

Esta estructura guarda los operadores aritméticos o lógicos que se van encontrando y los va apilando.

Stack de operandos y Stack tipos

La estructura operandos guarda las diferentes “variables” que va encontrando el lexer, mientras que el Stack tipos guarda el tipo de variable al paralelo de la pila anterior.

Funciones

enterExpFinal, enterTerminoFinal y enterExpresionFinal

Estas funciones detectan los diferentes tipos de operadores y los van almacenando en el Stack de operadores.

enterId_cte

Esta función detecta variables y constantes. Si detecta una variable verifica que esté declarada en la función actual o en la global y si detecta una constante la asigna a un tipo int o un tipo float.

generarCuadruploBinario

En este método se generan los cuádruplos para todos los operadores con excepción del de asignación. Para cumplir esta tarea se obtiene un operador y sus dos operandos, se verifica que sea una operación válida(con el cubo semántico), obtiene el temporal adecuado y se agrega al Stack operandos y Sack tipos.

exitExpFinal, exitTerminoFinal y exitExpresionFinal

Determinan cuando hay suficientes operandos y operadores para crear un cuádruplo y mandan a llamar a la función anterior.

exitAssign

En esta parte del código se crean los cuádruplos para el operador de asignación, este es diferente ya que solo usa un operando. Por ejemplo para $x = 1$ se genera el siguiente cuádruplo

Cuadruplo("=", 1, -1, x)

Generación de Cuadruplos Cíclicos y Condicionales

Para esta entrega se generan nuevos operadores que van a ayudar con la lógica de la máquina virtual en futuras entregas estos son GOTO, GOTOF, GOTOT. Estos

operadores me van a ayudar para poder saltar de un cuádruplo a otro, lo que será de gran utilidad para poder hacer saltos cíclicos o condicionales.

Estructuras

Stack pilaSaltos

Guarda los índices de los cuádruplos incompletos para GOTO, GOTOF y GOTOT, ya que estos solo pueden ser completados una vez que ya se haya encontrado a qué punto van a saltar.

Funciones

exitExpresion

Esta función se activa cuando detecta una condicional dentro de un if o un while y genera el cuádruplo GOTOF con el resultado de esa condicional y agrega el índice a la pilaSaltos.

exitCondition

Aquí existen múltiples caminos que puede tomar una condicional, el primero es que exista un else al terminar el if, en este caso genera el GOTO para saltar el else, saca el último elemento de la pilaSaltos y actualiza el GOTOF y guarda el índice del GOTO para poder completarlo al terminar la expresión. Si no hay un else solamente actualiza el salto del GOTOF original.

exitElsePart

Completa el GOTO de la función anterior cuando hay un else en la condicional.

enterCycle

Guarda el índice del inicio del ciclo en la pilaSaltos para ser usado al final por el cuádruplo GOTO.

exitCycle

Cuando termina el ciclo genera un GOTO con el índice guardado al inicio del ciclo y llena el GOTOF para poder salir cuando la expresión ya no sea verdadera.

Direcciones Asignadas

global_int: 1000-2999
global_float: 3000-4999
local_int: 5000-6999
local_float: 7000-8999
temp_int: 9000-10999

Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

temp_float: 11000-12999
temp_bool: 13000-14999
cte_int: 15000-16999
cte_float: 17000-18999
cte_string: 19000-20999

Para esta entrega también implementé las direcciones de memoria. Aunque hubo pequeños cambios en todo el código (principalmente cambiar argumentos de Strings a Integers), el mayor cambio fue la creación de la estructura Memoria. Esta se compone de un hashmap de strings e integers, el string define que tipo de variable es y el integer es el contador actual, esta implementación me permite tener un control de los contadores de todos los tipos de variables en un mismo lugar y con la creación de un método que permite agregar una nueva dirección al tipo de variable especificado la clase Memoria está lista para usarse con el código anteriormente propuesto. También se agregó una función que obtiene la cantidad de variables por tipo de variable y una que resetea a sus valores iniciales las variables temporales y locales. Finalmente se hace la creación de una tabla de constantes, que almacena todas las constantes encontradas y validadas.

Generación de Cuadruplos para Módulos

En esta entrega se generan nuevos operadores que me permiten declarar funciones y hacer llamadas con argumentos. Los nuevos operadores son los siguientes:

- ERA: Reserva la memoria.
- PARAM: Pasar parámetros de reales a formales.
- GOSUB: Saltar a la ejecución de una función.
- ENDF: Señalar el final de una función.

Estructuras

Para esta entrega se modificó la estructura de Funciones, se le agregó contadores para definir el espacio necesario para ejecutar la función y un índice de inicio, que define el cuádruplo donde va a dar inicio a la función.

Funciones

exitF_call

Se genera el cuádruplo ERA con la función actual. Itera sobre los parámetros de la función, revisa que se estén mandando el tipo correcto de variables y crea un cuádruplo PARAM por cada parámetro. Finalmente crea el cuádruplo GOSUB con el nombre de la función.

Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

exitFunc
Esta función se ejecuta al salir de la declaración de una función, se obtienen los contadores para las variables temporales y locales, se almacena la cantidad de variables necesarias para su ejecución y se crea el cuádruplo ENDF. Finalmente se resetea el contador de variables locales y temporales.

Maquina Virtual

Estructuras

HashMap MemoriaGlobal
Representa la memoria global, contiene todas las variables declaradas globalmente.

HashMap memoriaConstantes
Contiene las constantes del programa, pueden ser accedidas por su dirección.

Stack pilaMemoriaLocal
Pila de memorias locales activas, se usa para poder restaurar el contexto actual al hacer un ENDF (al terminar una función).

Stack pilaMemoriaTemporal
Pila que guarda las memorias temporales asociadas a las diferentes funciones, se usa junto a la memoria local para manejar la pila de ejecución.

Stack pilaRetorno
Este stack guarda la dirección de instrucciones donde debe continuar la ejecución después de terminar una función (GOSUB y ENDF).

HashMap memoriaLocalActual y memoriaTempActual
Memorias en uso actualmente para variables locales y temporales.

HashMap hashToNombreFuncion
Traduce el nombre de función ya que en los cuádruplos fueron almacenados como Integers hasheados.

Funciones por tipo de instrucción

Operadores Aritméticos (+, -, *, /)

Ejecuta el operador indicado y guarda el resultado en la dirección especificada por el cuádruplo. Usa doubleValue() para generalizar entre int y float.

Operadores Condicionales (<, >, !=)
Ejecuta la condicional y guarda el resultado como un booleano en la dirección indicada.

ObtenerValor
Lee el valor de la memoria correspondiente a la dirección indicada. Se usa en los operadores.

GuardarValor
Guarda un valor en la memoria correspondiente, según el rango de la dirección.

GOTOS

- GOTO: Salta incondicionalmente a resultado.
- GOTOF: Si operando1 es falso, salta a resultado; si es verdadero, continúa.
- GOTOV: Lo inverso a GOTOF.

ERA
Crea memorias temporales y locales con los valores guardados en la función durante la creación de cuádruplos.

CrearMemoria
Inicializa espacios de memoria según la dirección indicada. Se usa en ERA para poder crear los espacios necesarios para llamar a la función y al inicio del programa para crear los espacios globales.

PARAM
Toma un valor ya evaluado y lo copia a la dirección esperada del parámetro.

GOSUB
Guarda el contexto actual a la pila de memorias y salta al inicio de la función.

ENDF
Restaura el contexto previo desde las pilas (temporal, local e ip).

Código en Consola de Ejecución

```
mvn exec:java -Dexec.mainClass="com.ok.Main2"
```

Oskar Arturo Gamboa Reyes
A01173648
02 de junio de 2025
BabyDuck - Entrega Final

Link GitHub:
https://github.com/oskgamart/Compiladores/tree/main/Scanners%20y%20Parsers/duck