Oskar Arturo Gamboa Reyes
A01173648
29 de abril de 2025
BabyDuck - Entrega #1

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 ;

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

Oskar Arturo Gamboa Reyes
A01173648
29 de abril de 2025
BabyDuck - Entrega #1

```
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
               | ;
```

Oskar Arturo Gamboa Reyes
A01173648
29 de abril de 2025
BabyDuck - Entrega #1

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
29 de abril de 2025
BabyDuck - Entrega #1

```
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
29 de abril de 2025
BabyDuck - Entrega #1

```
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
```

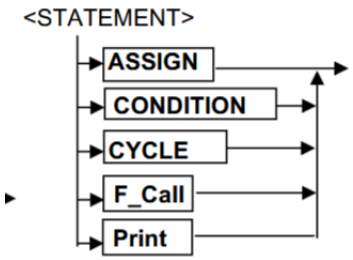
Oskar Arturo Gamboa Reyes
A01173648
29 de abril de 2025
BabyDuck - Entrega #1

```
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'
```

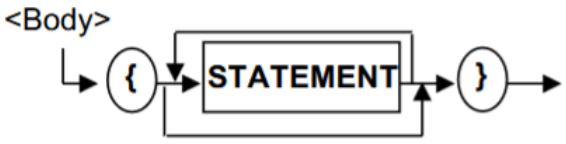
Comentarios

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
29 de abril de 2025
BabyDuck - Entrega #1

```
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.

Link GitHub:
https://github.com/oskgamart/Compiladores/tree/4f2925f970a76497a308e186856e763df47a3c59/Scanners%20y%20Parsers