Baby Duck

Valeria Enríquez Limón

A00832782

Submitted in partial satisfaction of the requirements for the degree of

BSc (Ingeniería en Tecnologías Computacionales) Desarrollo de aplicaciones avanzadas de ciencias computacionales (Gpo 503)

Instituto Tecnológico de Estudios Superiores de Monterrey

ITESM

2025

Table of Contents

List of Tables

Chapter 1

Análisis Léxico y Gramatical del Lenguaje BabyDuck: Expresiones Regulares, Tokens y CFG

En la primera entrega se realizo el Análisis Léxico y Gramatical del Lenguaje BabyDuck: Expresiones Regulares, Tokens y CFG.

1.1 Reglas Gramaticales

Las reglas gramaticales son un conjunto de producciones que definen la estructura sintáctica de un lenguaje formal. A diferencia de las expresiones regulares, que se enfocan en el análisis léxico, las reglas gramaticales permiten describir cómo se combinan los símbolos y palabras para formar frases válidas dentro del lenguaje, es decir, su sintaxis.

Estas reglas suelen representarse mediante gramáticas libres de contexto (CFG, por sus siglas en inglés), las cuales son ampliamente utilizadas en el diseño de lenguajes de programación y en la construcción de analizadores sintácticos. Una gramática está compuesta por un conjunto de símbolos terminales, no terminales, un símbolo inicial y un conjunto de producciones que especifican cómo los no terminales pueden transformarse.

En esta sección se presentan las principales reglas gramaticales del lenguaje en estudio, así como su notación y análisis estructural.

1. $c\ te \rightarrow c\ te$ - int
   $c\ te \rightarrow c\ te$ - float
2. type $\rightarrow$ int
   type $\rightarrow$ float
3. EXP $\rightarrow$ TEMINO E'
   op $\rightarrow +$
   op $\rightarrow -$
   $E \rightarrow$ OP EXP
   $E \rightarrow \varepsilon$
4. TEMINO $\rightarrow$ FACTOR E'
   op $\rightarrow *$
   op $\rightarrow /$
   $E \rightarrow$ op TEMINO
   $E \rightarrow \varepsilon$
5. Statement $\rightarrow$ assign
   Statement $\rightarrow$ condition
   Statement $\rightarrow$ cycle
   Statement $\rightarrow$ #-call
   Statement $\rightarrow$ Print
6. FUNCS $\rightarrow$ void id (F') [V' Body] ?
   F' $\rightarrow$ id : TYPE F" F"'
   F" $\rightarrow \varepsilon$
   F" $\rightarrow$,
   F"' $\rightarrow \varepsilon$
   F"' $\rightarrow$ id : TYPE F" F"'

V' $\rightarrow$ ε

V' $\rightarrow$ VARS V'

7. Body $\rightarrow$ $\{S'\}$

   s' $\rightarrow$ STATEMENTS

   s' $\rightarrow$ ε

8. ASSIGN $\rightarrow$ id = EXPRESSION

9. PRINT $\rightarrow$ print (P') ;

   P' $\rightarrow$ EXPRESSION P"

   P" $\rightarrow$ ε

   P" $\rightarrow$, C'

   C' $\rightarrow$ ε

   C' $\rightarrow$ cte.strings

10. CONDITION $\rightarrow$ if (EXPRESSION) BODY C'

    C' $\rightarrow$ ε

    C' $\rightarrow$ else Body

11. Expression $\rightarrow$ EXP E' EXP

    E' $\rightarrow$>|<|! =|=

    E' $\rightarrow$ ε

12. TERM $\rightarrow$ id | c.te | (EXPRESSION)

13. programa $\rightarrow$ program id, PP' main Body end

    P $\rightarrow$ ε

    P $\rightarrow$ VARS

P' → ε

P → FUNCS P'

14. VARS → var id V' : TYPE ;

    V' → , id V'

    V' → ε

    V" → ε

    V" → id V' : TYPE ; V"

1.2 Expresiones Regulares

Las expresiones regulares son una herramienta fundamental en el análisis sintáctico de lenguajes formales, utilizadas para describir patrones dentro de cadenas de texto. En el contexto de los lenguajes de programación y los compiladores, permiten definir de manera concisa y precisa los elementos léxicos que forman parte de un lenguaje, como identificadores, operadores, números y palabras clave.

Es importante destacar que las expresiones regulares forman parte de los lenguajes regulares, los cuales pueden ser representados también mediante autómatas finitos. Esta correspondencia permite que las expresiones regulares sean implementadas eficientemente en analizadores léxicos, convirtiéndose en el primer paso dentro del proceso de compilación.

A continuación, se presentan las principales definiciones, operadores y reglas que rigen el uso de expresiones regulares en este contexto.

Token	**Lexema(s)**	**Expresión Regular**
PROGRAM	`program`	`program`
ID	Identificador	`[a-zA-Z_][a-zA-Z0-9_]*`
SEMICOLON	`;`	`;`
COLON	`:`	`:`
COMMA	`,`	`,`
LPAREN	`(`	`\(`
RPAREN	`)`	`\)`
LBRACE	`{`	`{`
RBRACE	`}`	`}`
MAIN	`main`	`main`
END	`end`	`end`
VAR	`var`	`var`
TYPE	`int`, `float`	`int|float`
VOID	`void`	`void`
ASSIGN	`=`	`=`
EQ	`==`	`==`
NEQ	`!=`	`!=`
LT	`<`	`<`
GT	`>`	`>`
PLUS	`+`	`\+`
MINUS	`-`	`-`
MULT	`*`	`\*`
DIV	`/`	`/`
IF	`if`	`if`
ELSE	`else`	`else`
WHILE	`while`	`while`
DO	`do`	`do`
PRINT	`print`	`print`
CTE_INT	Constante entera	`[0-9]+`
CTE_FLOAT	Constante flotante	`[0-9]+\.[0-9]+`

Table 1.1: Tokens del lenguaje `Baby_Duck` y sus expresiones regulares