

**Diseño de Compiladores (Gpo 2)**

**Documentación del Compilador**

**Luis Eugenio Candelaria Azpilcueta (ITC)**

**A00816826**

**29 de septiembre de 2021 Monterrey, Nuevo León**

**Expresiones Regulares**

De acuerdo al documento de requerimientos se llegó a una lista de tokens y palabras reservadas en el lenguaje, cabe recalcar que hay una serie de palabras especiales que no suele tener cualquier lenguaje pero se necesitan en este compilador, tales como la media, moda, varianza, etc.

La lista de tokens es la siguiente (la columna de la derecha no está basada en lo que está en código en al analizadorLexico.py):

|  |  |
| --- | --- |
|  |  |
| Token | **¿Cómo se lee?** |
| ID | [a-zA-Z\_][a-zA-Z\_0-9] |
| CTE\_STRING | [a-zA-z\_] |
| CTE\_INT | [0-9\_] |
| CTE\_FLOAT | [0-9]+.[0-9] |
| CTE\_CHAR | [a, b, c,…] o [A, B, C,…] |
| COLON | : |
| SEMICOLON | ; |
| COMA | , |
| LBRACK | { |
| RBRACK | } |
| LPAREN | ( |
| RPAREN | ) |
| LSBRACK | [ |
| RSBRACK | ] |
| DIFF | <> |
| EQUAL | == |
| GTHAN | > |
| LTHAN | < |
| PLUS | + |
| MINUS | - |
| TIMES | \* |
| DIVIDE | / |
| ASSIGN | = |
| AND | & |
| OR | | |

En cuanto a las palabras reservadas, son las siguientes:

|  |
| --- |
| Palabras reservadas |
| IF |
| THEN |
| ELSE |
| PROGRAM |
| MAIN |
| VARS |
| INT |
| FLOAT |
| CHAR |
| FUNCTION |
| VOID |
| RETURN |
| READ |
| WRITE |
| WHILE |
| DO |
| FOR |
| TO |
| AVERAGE |
| MODE |
| VARIANCE |
| REGRESSION |
| PLOTXY |

**Gramática**

Para la gramática tuve que considerar una serie de cosas antes de entrar de lleno al esto que es el diseño de mi compilador, aún cuando la sintáxis es solo una parte a analizar del compilador, la complejidad con la que lo quería diseñar me exijía que el analizador sintáctico fuera como una especie de puerta a todo lo demás, ¿qué quiero decir con esto?, las reglas de sintaxis pueden ser claras pero como usuario hay muchas formas de evadirlas; para aclarar este punto, hablaré específicamente de una regla, en las especificaciones se pide el usuario pueda y solamente de esa forma, declarar variables que sean globales, no es como en C++, Python o cualquier otro lenguaje popular que te permite declarar variables dentro de un bloque, ¿cuál es el problema?, de la forma que yo lo veo, es que cuando quiero hacer referencia a una variable arreglo del tipo que sea, quisiera usar un indice en base a una operación matemática, como lo es el clásico:



En el ejemplo anterior vemos cómo el índice no es precisamente una operación matemática, pero sí es derivada de las reglas sobre expresiones, en otras palabras, una regla de qué poner dentro de los corchetes sería permitirle al usuario poner una exresión, y de ahí llegar hasta un ID como una regla derivada de las expresiones. El conflicto que uno tiene con este punto es que en variables globales, poner una regla de expresión genérica sería absurdo, por el camino que esta claramente señalado, en una variable arreglo global no puedes darle como tamaño una operación que hace referencia a un ID, porque el mismo no tiene ningún valor para ese punto, y la misma incongruencia se encuentra con otras reglas derivadas de expresiones, como lo es el poder usar una compuerta lógica OR o AND. Para tratar este problema decidí crear una regla de expresiones específica para la declaración de variables, así cuando se quiera crear un arreglo global, se puede determinar su tamaño con una suma, resta, multiplicación y división, pero solo con números enteros, evadiendo las demás reglas derivadas de la de expresión genérica.

El usar esta metología creará una cantidad considerable de reglas de sintáxis, y se aprovechó esta situación creando un analizador semántico, que tiene como propósito tener un resultado visual y textual de lo que el usuario ha escrito en el lenguaje que representa el compilador. Más detalles se hablarán sobre el analizador semántico en un futuro capítulo. Solo como adelanto, el contenido resultante del analizador semántico permite ir paso a paso en el diagrama de sintaxis creado en cada caso de prueba, con esto, desde el mismo analizador de sintaxis se puede analizar todo lo demás, desde variables globales, hasta llegar a los cuádruplos.

Como ya he mencionado las reglas de sintaxis son bastantes y muy espcíficas, pero puedo asegurar que con estas los errores por parte del compilador son escasos. Las reglas son las siguientes:

program -> PROGRAM ID SEMICOLON programVer1

PROGRAM ID SEMICOLON programVer2

PROGRAM ID SEMICOLON programVer3

programVer1 -> variables functions principal

programVer2 -> variables principal

functions principal

programVer3 -> principal

variables -> VARS varAux

varAux -> oneVar

sameType

newType

oneVar -> type COLON identifierVar SEMICOLON

sameType -> type COLON identifierVar COMA sameTypeFinal

type COLON identifierVar COMA sameTypeRecursive

sameTypeFinal -> identifierVar SEMICOLON

sameTypeRecursive -> identifierVar COMA sameTypeRecursive

identifierVar COMA sameTypeFinal

identifierVar SEMICOLON varAux

newType -> type COLON identifierVar SEMICOLON varAux

functions -> funcAux

recursiveFunc

recursiveFunc -> funcAux functions

funcAux -> FUNCTION typeFunction ID LPAREN withParemeters

FUNCTION typeFunction ID LPAREN RPAREN funcVer1

FUNCTION typeFunction ID LPAREN withParemeters

FUNCTION typeFunction ID LPAREN RPAREN funcVer2

typeFunction -> INT

FLOAT

CHAR

VOID

withParameters -> parameters RPAREN funcVer1

parameters RPAERN funcVer2

funcVer1 -> variablesLoc block

funcVer2 -> block

variablesLoc -> VARS varAuxLoc

varAuxLoc -> oneVarLoc

sameTypeLoc

newTypeLoc

oneVarLoc -> type COLON identifier SEMICOLON

sameTypeLoc -> type COLON identifier COMA sameTypeFinalLoc

type COLON identifier COMA sameTypeRecursiveLoc

sameTypeFinalLoc -> identifier SEMICOLON

sameTypeRecursiveLoc -> identifier COMA sameTypeRecursiveLoc

identifier COMA sameTypeFinalLoc

identifier SEMICOLON varAuxLoc

newTypeLoc -> type COLON identifier SEMICOLON varAuxLoc

parameters -> oneParam

sameTypeParam

newParam

oneParam -> type COLON identifierVar

sameTypeParam -> type COLON identifierVar COMA sameTypeParamFinal

type COLON identifierVar COMA sameTypeParamRecursive

sameTypeParamFinal -> identifierVar

sameTypeParamRecursive -> identifierVar COMA sameTypeParamRecursive

identifierVar COMA sameTypeFinal

identifierVar SEMICOLON parameters

newParam -> type COLON identifierVar SEMICOLON parameters

identifierVar -> indentLonelyVar

identArrayVar

identLonelyVar -> ID

identArrayVar -> ID LSBRACK expressionVar RSBRACK

type -> INT

FLOAT

CHAR

principal -> MAIN LPAREN RPAREN block

block -> LBRACK blockAux

blockAux -> statute RBRACK

blockRecursive

blockFinal

blockFinal -> RBRACK

blockRecursive -> statute blockAux

statute -> assignment SEMICOLON

if\_statement

writing SEMICOLON

reading SEMICOLON

loop

functionCall SEMICOLON

return\_statement SEMICOLON

regressionFunc SEMICOLON

plotXYFunc SEMICOLON

assignment -> identifier ASSIGN expression

plotXYFunc -> PLOTXY LPAREN ID COMA ID RPAREN

regressionFunc ->

return\_statement -> RETURN LPAREN expression RPAREN

if\_statement -> shortCondition

longCondition

shortCondition -> IF LPAREN expression RPAREN THEN block

longCondition -> IF LPAREN expression RPAREN THEN block ELSE block

writing -> WRITE LPAREN writingAux

writingAux -> writingFinal

writingRecursive

writingFinal -> CTE\_STRING RPAREN

expression RPAREN

writingRecursive -> CTE\_STRING COMA writingFinal

expression COMA writingFinal

CTE\_STRING COMA writingRecursive

expression COMA writingRecursive

reading -> READ LPAREN readingAux

readingAux -> readingFinal

readingRecursive

readingFinal -> identifier RPAREN

readingRecursive -> identifier COMA readingRecursive

identifier COMA readingFinal

loop -> conditional

nonconditional

conditional -> WHILE LPAREN expression RPAREN block

nonconditional -> FOR identifier ASSIGN expression to expression DO block

functionCall -> ID LPAREN callAux RPAREN

callAux -> expression

callRecursive

callRecursive -> expression COMA callRecursive

expression COMA expression

expression -> term

expressionOperation

expressionOperation -> term PLUS expression

term MINUS expression

term -> factor

termOperation

termOperation -> factor TIMES term

factor DIVIDE term

factor -> idioms

factorOperation

factorOperation -> idioms GTHAN factor

idioms LTHAN factor

idioms EQUAL factor

idioms DIFF factor

idioms -> clause

idiomsOperation

idiomsOperation -> auction OR idioms

auction AND idioms

auction -> constant

otherExpression

identifier

averageFunc

modeFunc

varianceFunc

functionCall

constant -> CTE\_INT

CTE\_FLOAT

CTE\_CHAR

otherExpression -> LPAREN expression RPAREN

identifier -> identLonely

identArray

identLonely -> ID

identArray -> ID LSBRACK expression RSBRACK

averageFunc -> AVERAGE LPAREN ID RPAREN

modeFunc -> MODE LPAREN ID RPAREN

varianceFunc -> VARIANCE LPAREN ID RPAREN

expressionVar -> termVar

expressionOperationVar

expressionOperationVar -> termVar PLUS expressionVar

termVar MINUS expressionVar

termVar -> auctionVar

termOperationVar

termOperationVar -> auctionVar TIMES termVar

auctionVar DIVIDE termVar

auctionVar -> constantVar

otherExpressionVar

constantVar -> CTE\_INT

otherExpressionVar -> LPAREN expressionVar RPAREN