**ANALIZADOR LÉXICO Y SINTÁCTICO DE UNA GRAMÁTICA ARITMÉTICA  
LEXICAL AND SYNTACTIC ANALYZER OF AN ARITHMETIC GRAMMAR**

Jhan Ignacio1, Jeyson Chávez2, Braulio Anchante3, Brayan Vargas4, Camila Ortiz5, Ronald Ruiz6, Laura Bazán7

1-6Estudiantes. Ingeniería de Sistemas Computacionales. Universidad Privada del Norte. Perú.

7Docente. Ingeniería de Sistemas Computacionales. Universidad Privada del Norte. Perú

**Resumen**

Para el presente trabajo se tiene como objetivo principal desarrollar e implementar un analizador léxico y sintáctico enfocado en operaciones aritméticas para una gramática de contexto libre. De esta manera, lograr manejar conceptos del analizador léxico y sintáctico, aprender sus funciones, conocer las estrategias de implementación de analizadores léxicos. Además, tener en cuenta conceptos básicos trabajados a lo largo del desarrollo del curso, tales como: analizador léxico y sintáctico, patrones, derivaciones, lexemas, tokens, entre otros. El alcance del presente trabajo tiene como finalidad el análisis léxico y sintáctico, implementados en software. Para el desarrollo se hace uso del programa ZinjaI vers. 2021 utilizando el lenguaje de programación C++.

**Palabras clave**: analizador léxico, analizador sintáctico, Tokens, Lexemas.

**Abstract**

The main objective of this work is to develop and implement a lexical and syntactic analyzer focused on arithmetic operations for a free context grammar. In this way manage to handle concepts of the lexical and syntactic analyzer, learn its functions, know the implementation strategies of lexical analyzers. In addition, take into account basic concepts worked throughout the course development, stories such as: lexical and syntactic analyzer, patterns, derivations, lexemes, tokens, etc. The scope of this work has as a method lexical and syntactic analysis, implemented in software. The ZinjaI vers.2021 program is used for development using the C ++ programming language.

**Keywords**: Lexical Analyzer, parser, Tokens, Lexemes.

1. **INTRODUCCIÓN**

Un compilador es una herramienta que permite interpretar lenguajes de alto nivel. Los componentes de un compilador son los analizadores léxico, sintáctico y semántico. La primera fase del análisis de un código fuente es el análisis léxico, el cual tiene como principal objetivo leer el flujo de caracteres de entrada y convertirlo en una serie de “*tokens*” o componentes léxicos, que son las unidades mínimas de cada lenguaje de programación y son estos los que usa el analizador sintáctico. El análisis léxico es la primera fase del compilador y el módulo principal encargado de la misma es el analizador léxico, también conocido como Lexer. La cual se encarga de transformar el programa fuente en una secuencia de tokens a partir de la lectura de dicho programa fuente carácter a carácter, también se encarga de la detección de errores léxicos; este tipo de error se detecta cuando el analizador léxico encuentra una secuencia de caracteres que no encaja con ninguno de los tokens [1] .

El analizador léxico va leyendo del fichero de entrada los caracteres secuencialmente y los van agrupando, según las directivas programadas, en tokens con un significado conocido por el programador, los tokens serán la entrada de analizador léxico. El analizador léxico hace las funciones, a la vez de preprocesador ya que se encarga de eliminar los caracteres innecesarios para el proceso de compilación, elimina los espacios en blanco que hay entre palabras [2].

Según Vega [3] un metacompilador es un compilador de compiladores. Se trata de un programa que acepta como entrada la descripción de un lenguaje y produce el compilador de dicho lenguaje.

Como antecedentes se ha tomado el trabajo de Villar [4] donde resume que el analizador léxico divide la entrada en componentes léxicos, de tal manera que los componentes se agrupan en categorías léxicas. Se especifican las categorías mediante expresiones regulares, haciendo el tratamiento de errores en el nivel léxico muy simple. Además, en el trabajo de Vásquez [5] un análisis sintáctico verifica si las frases formadas por los componentes léxicos tienen una estructura que se ajuste a las reglas especificadas para las sentencias del lenguaje (a su gramática)

Se desarrolló un software que comprueba el autómata e informa al usuario si existen errores, con el fin de ejecutar la aplicación sin problemas; si existe algún error, simplemente se detiene, ya que solo admite una gramática específica.

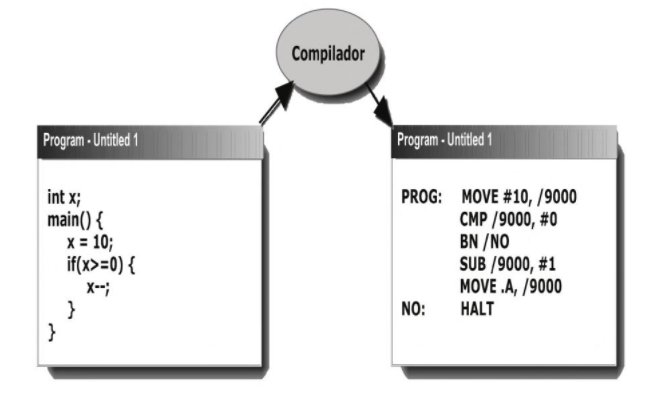
1. **MATERIALES Y MÉTODOS**

A lo largo de la elaboración del trabajo se hace uso de temas desarrollados durante el presente ciclo; en este apartado se va a detallar una definición más clara de los temas y términos importantes a utilizar los cuáles son: traductor o compilador, gramática, autómata, definición formal de un autómata finito, diagrama de transición, análisis léxico, tokens, lexema, patrón y espacios en blanco.

Para la elaboración del proyecto se hace uso de un entorno de desarrollo llamado ZinjaI vers. 2021 utilizando el lenguaje de programación C++.

* 1. **Compilador**

Un compilador es un tipo especial de traductor que interpreta un programa hecho en un lenguaje de alto nivel a un programa funcionalmente equivalente con un lenguaje de bajo nivel. Un traductor es cualquier programa que toma como entrada un texto escrito en un lenguaje, llamado fuente y da como salida otro texto en un lenguaje, denominado objeto [6]. El compilador analiza todo el programa fuente, crea el programa objeto y luego permite su ejecución. Debido a que es un programa complejo consta de una serie de pasos, generalmente entrelazados, los cuales están bien definidos y en cierta medida sistematizados, aunque no están faltos de dificultad lo que aumenta conforme se incrementa la riqueza del lenguaje a compilar [7].



**Fig. 1.** Esquema de un Compilador

* 1. **Gramática**

Es el conjunto de reglas las cuales indican si las sentencias construidas a partir de las palabras pertenecen o no al lenguaje [8]. El uso de gramáticas es otra forma de describir un lenguaje en forma general y rigurosa [9].

La notación de Chomsky define una gramática como una tupla (N, T, P, S)

* N es el conjunto de los no terminales (conjunto de símbolos que introducimos en la gramática como elementos auxiliares que no forman parte de las sentencias del lenguaje).
* T es el conjunto de los terminales (son los símbolos o caracteres que forman las sentencias del lenguaje).
* P es el conjunto de reglas de producción (reglas de derivación de las tiras).
* S es el axioma de la gramática, S está incluido en N (es el símbolo inicial o metanoción).
  1. **Autómata**

Un autómata es una máquina teórica que lee instrucciones en forma de símbolos y cambia de estado según éstas. Acepta una información de entada (input), la procesa sometiéndola a transformaciones simbólicas que pueden adoptar la forma de un cálculo o computación generando un resultado o salida (output). Existen distintas áreas de aplicación para los autómatas las cuales son; compiladores, comunicaciones, teoría de control, circuitos secuenciales, reconocimiento de patrones [9].

* 1. **Autómata Finito**

Es un mecanismo cuya memoria es siempre finita. A lo largo del proceso los contenidos de memoria del autómata van cambiando, así mismo se produce el cambio de estado definiendo una función de transición de estados, que nos dice cual será el estado del autómata en el instante de tiempo siguiente, dicha función de transición de estados puede ser determinista (AFD) o no determinista (AFND) [10].

Un autómata finito se puede definir como una tupla de la forma:

AF = (Q, Te, δ, q0, F)

Q es el conjunto de estados.

Te es el alfabeto de entrada.

q0 es el estado inicial.

F es el conjunto de estados finales (F ⊂ Q).

δ es la función: Q x {Te U {λ}} → P(Q) [8]

* 1. **Diagrama de transición**

Generalmente se asocia con cada autómata un grafo dirigido, llamado diagrama de transición de estados. Cada nodo del grafo corresponde a un estado. El estado inicial se indica mediante una flecha que no tiene nodo origen. Los estados finales se representan con un círculo doble [11]. Cada transición de estados tiene un arco asociado etiquetado con el/los símbolos correspondientes.

Llamamos movimiento a la transición de un estado a otro y se representa de la siguiente forma:

(q, aα) |---- (q´, α)

q´ = δ(q, a), en el caso de un AFD q´ sería el único estado al que podría transitar, en el caso de un AFND existirían varios estados, es decir, δ(q, a) sería un conjunto.

* 1. **Análisis Léxico (Scanner)**

También denominado análisis lineal, o exploración. Dicha fase de la compilación se encarga de verificar si todas las cadenas pertenecen o no al lenguaje. Realiza un análisis símbolo por símbolo indicando el token por cada uno de los elementos reconocidos o el error en caso de no reconocer. En un programa fuente existen una serie de símbolos con los cuales se representan las construcciones del lenguaje tales como variables, etiquetas, palabras reservadas, constantes,

operadores, etc. Es necesario que el traductor identifique los distintos significados de estas construcciones. El programa fuente se trata inicialmente con el analizador léxico (en inglés scanner) con el propósito de agrupar el texto en grupos de caracteres con entidad propia llamados tokens [6].

* 1. **Tokens**

Son unidades sintácticas o componentes léxicos, tales como constantes, identificadores (de variables, de funciones, de procedimientos, de tipos, de clases), palabras reservadas, y operadores. Por razones de eficiencia, a cada token se le asocia un atributo (o más de uno) que se representa internamente por un código numérico o por un tipo enumerado [6]. Pueden ser de dos tipos; cadenas específicas como palabras reservadas, puntos y comas, etc., y no específicas como identificadores, contantes y etiquetas.

En la fase de análisis léxico se lee el texto del código fuente carácter por carácter y se van generando los tokens. Estos tokens constituyen la entrada para el siguiente proceso de análisis (análisis sintáctico). El agrupamiento de caracteres en tokens depende del lenguaje que vamos a compilar [7].

* 1. **Lexema**

El analizador léxico lee los caracteres que componen el texto del programa fuente y suministra tokens al analizador sintáctico. El analizador léxico irá ignorando las partes no esenciales para la siguiente fase, como pueden ser los comentarios o espacios en blanco. Los tokens se consideran entidades con dos partes: su tipo y su valor o lexema. En algunos casos, no hay tipo (como en las palabras reservadas). Si tenemos una variable “x”, su tipo es identificador y su lexema es x [7].

Es una secuencia de caracteres en el programa fuente, que coinciden con el patrón para un token y que el analizador léxico identifica como una instancia de ese token [12].

* 1. **Zinjai**

Es un entorno de desarrollo integrado libre y gratuito para programar en C/C++. Pensado originalmente para ser utilizado por estudiantes de programación durante el aprendizaje, presenta una interfaz inicial muy sencilla, pero sin dejar de incluir funcionalidades avanzadas que permiten el desarrollo de proyectos complejos [13].

**2.10 Lenguaje C ++**

En 1972 se crea el lenguaje C como el lenguaje para el desarrollo de programas en los sistemas UNIX. A principios de los 80, Bjarne Stroustrup creó el lenguaje C++ como una extensión de C y evolucionó hasta el estándar de 1998. El nuevo lenguaje C++ no es una simple modificación de algunos aspectos de C, sino que además se le ha dotado de nuevas posibilidades, destacando sobre todo la programación orientada a objetos [14].

Para el desarrollo del software se tiene los siguientes tipos de componentes léxicos o tokens:

- **Identificadores**, que sólo son nombres de variables y están compuestos por una letra minúscula de rango a..z.

- **Constantes numéricas** de un dígito, de rango 0..9.

- **Operadores**: +, -, \*, / y %.

- **Símbolo de asignación**: = (igual).

- **Paréntesis**: ( y ).

- **Separador de sentencias**: ; (punto y coma).

- **Indicadores de principio y fin de bloque**: { y }.

- **Palabras reservadas** : R (Read), W (Write) y M (Main).

Se realizará formalmente mediante una gramática EBNF, que es LL(1), es decir el analizador sintáctico conoce qué regla de producción ha de seguir con sólo conocer el token por el que comienza la sentencia.

La gramática en EBNF del lenguaje MUSIM/0 es:

<programa> ::= M {<bloque> }

<bloque> ::= <sentencia> <otra\_sentencia>

<otra\_sentencia> ::= ; <sentencia> <otra\_sentencia> | <vacío>

<sentencia> ::= <asignación>; | <lectura>; | <escritura>;

<asignación> ::= <variable> = <expresión>

<expresión> ::= <término> <más términos>

<más términos> ::= + <término> <más términos>| - <término> <más términos>| <vacío>

<término> ::= <factor> <más factores>

<más factores>::= \* <factor> <más factores>| / <factor> <más factores>| % <factor> <más factores> | <vacío>

<factor> ::= ( <expresión> ) | <variable> | <constante>

<lectura> ::= R <variable>

<escritura> ::= W<variable>

<variable> ::= (a | b | c | ... | z)

<constante> ::= (0 | 1 | ... | 9)

<vacío> ::=

Puede observarse que este lenguaje sólo permite tres tipos de sentencias: lectura, asignación y escritura. Tiene un solo tipo de datos: entero. Tiene cinco operadores + (adición), - (diferencia), \* (producto), / (división entera), y % (módulo). Se permite el uso de paréntesis. Atención, no se permite menos unario.

Ejemplos de programas a utilizar:

1)

M { R a; R b; c = a + b - 2 ; W c }

2)

M {R a ; R b; R c; p = (a + b + c ) / 2 ;W p }

3)

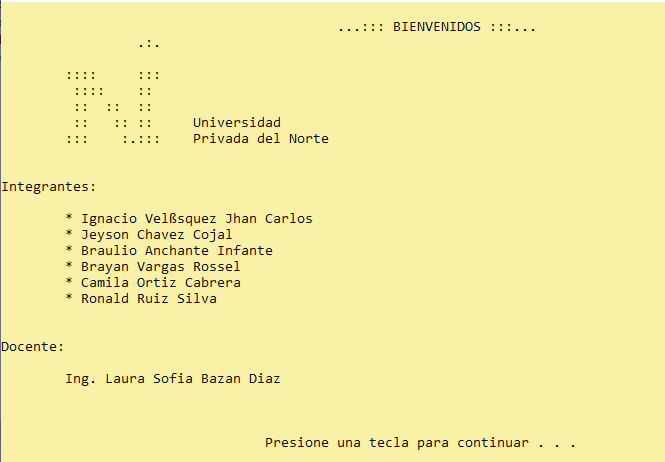
M{R a ;R b ;R c ;R d ;R e ;f = ( ( a + 5 ) \* ( b - c ) ) % ( d / ( 8 - e ) ) ;W f}

4)

M{ R x; R y; d = x \* x + y \* y; i = x \* x - y \* y;c = (d + x) / (d -y) ; W d; W i; W c}

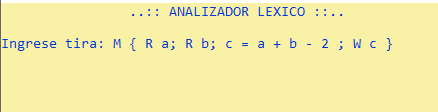
1. **RESULTADOS**

En la figura 2 vemos la pantalla de inicio del programa.



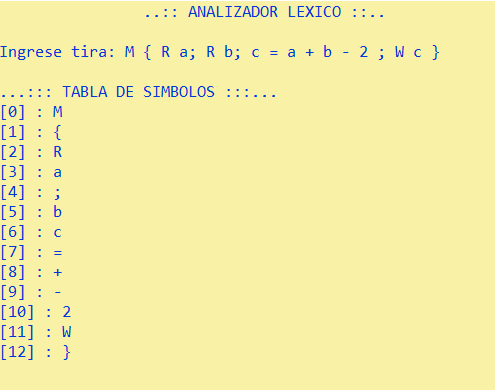
**Fig. 2.** Pantalla de inicio

En la figura 3, se ingresa los caracteres que van a ser analizados según la gramática dada.



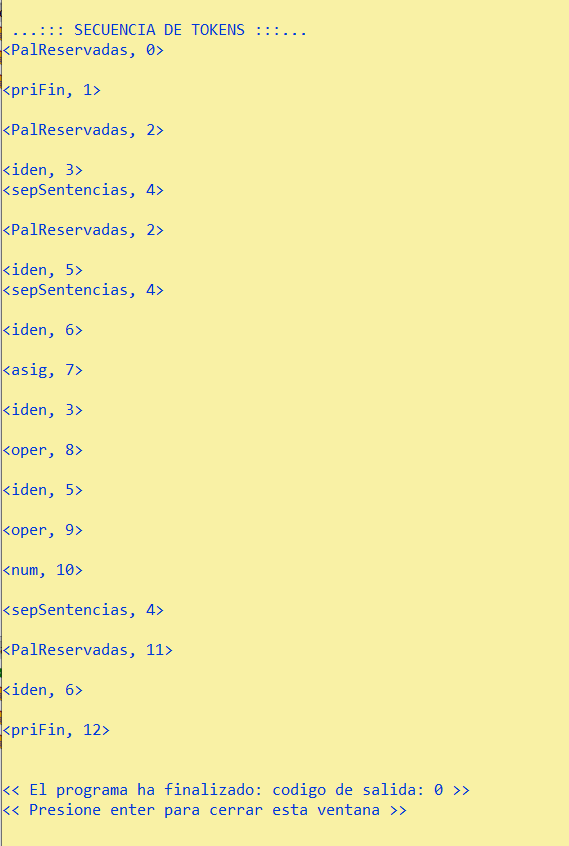
**Fig. 3.** Ingreso de una tira

En la figura 4 se observa la tabla de símbolos obtenidos por el analizador léxico implementado.



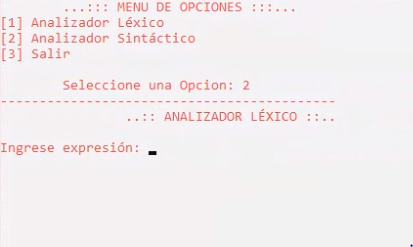
**Fig. 4.** Tabla de símbolos

En la figura 5 se observa la secuencia de tokens obtenidos a partir de la gramática ingresada.



**Fig.5.** Secuencia de Tokens

En la figura 6 se observa la implementación de un menú de opciones que indica el número a ingresar para elegir una opción.



**Fig.6.** Menú de opciones

1. **DISCUSIÓN**

A partir de los resultados encontrados, la mayoría de las expresiones ingresadas devuelven una secuencia de componentes léxicos y el análisis sintáctico. Por el contrario, cuando se ingresa una expresión de manera incorrecta devuelve el error que se está cometiendo. Estos resultados guardan relación con lo que sostiene JM Villar Torres (2009), pues las expresiones se agrupan en categorías léxicas, haciendo el tratamiento de errores en el nivel léxico muy simple. Ello es acorde con lo que en este trabajo se halla.

1. **CONCLUSIONES**

Se pudo definir el uso de los analizadores léxicos para una gramática aritmética devolviendo el resultado de una cadena de caracteres de entrada, la tabla de símbolos, secuencia de token y el parse izquierdo.

Se verificó y se resolvió el problema de congruencia gracias a los analizadores léxicos.

Así mismo se dio un enfoque más claro de la importancia de lexemas, patrón, tokens y parse, para tener un mejor entendimiento en cuanto a temas desarrollados en el curso de Compiladores y Lenguajes de Programación.

Finalmente se logró conocer las estrategias de implementación para un analizador léxico y sintáctico para una gramática aritmética.

1. **RECOMENDACIONES**

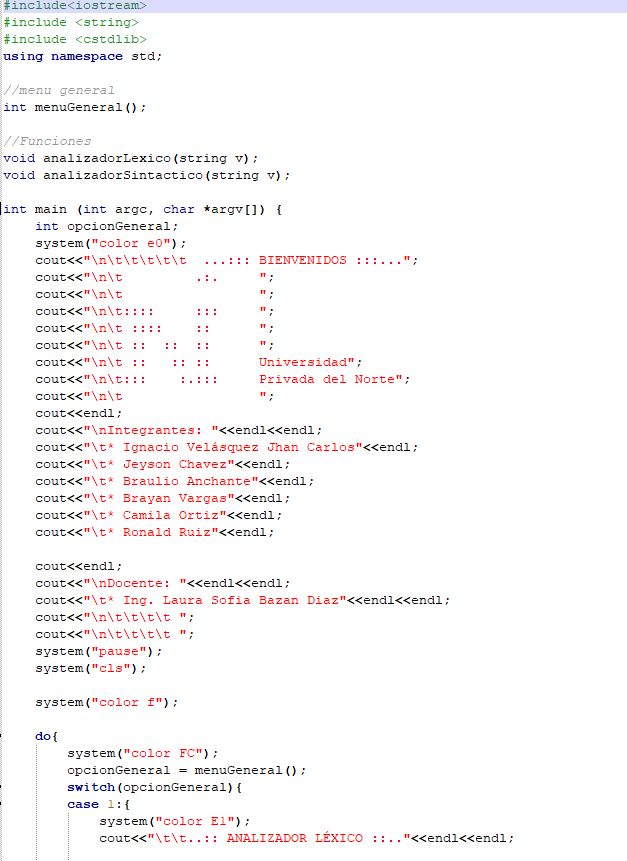
Basándonos en las conclusiones se recomienda que para la elaboración de un analizador léxico y sintáctico es necesario tener en cuenta las investigaciones anteriores para que sirva como base para la elaboración del analizador. Se recomienda conocer conceptos básicos para la buena estructuración de analizador léxico y sintáctico. Para así obtener buenos resultados en la construcción del programa.

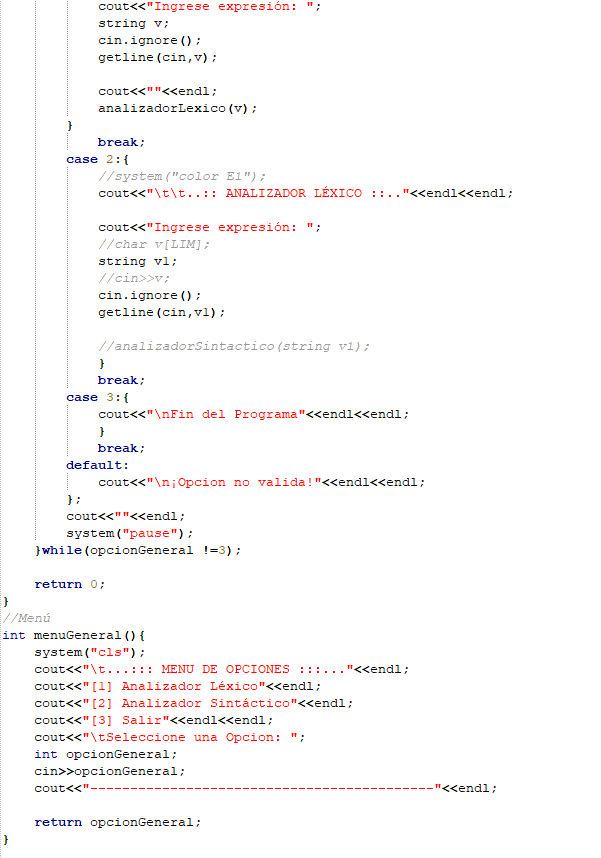
# **REFERENCIAS BIBLIOGRAFICAS**

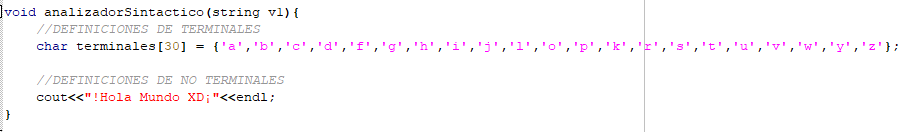
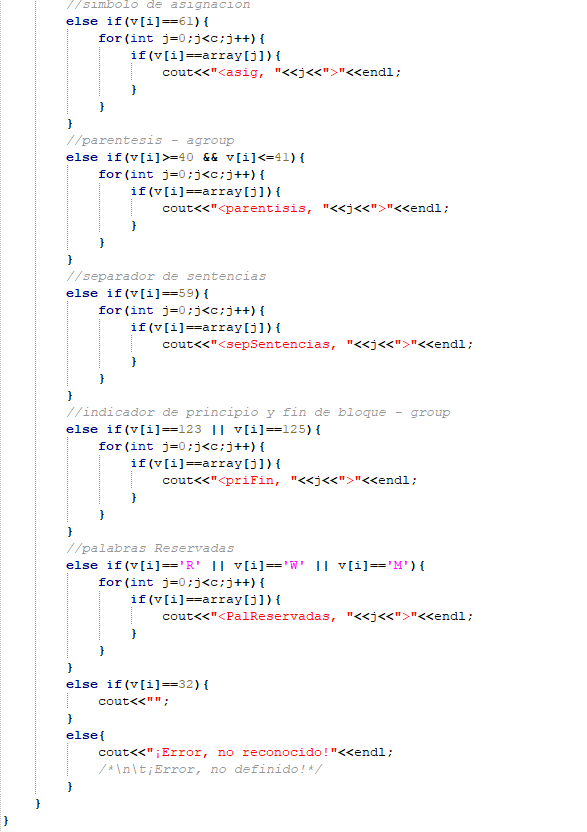
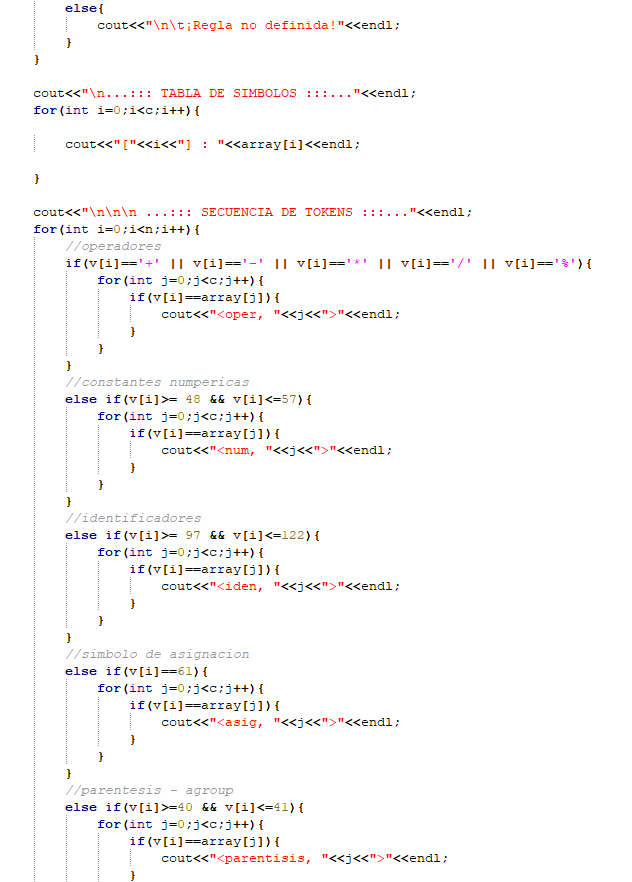
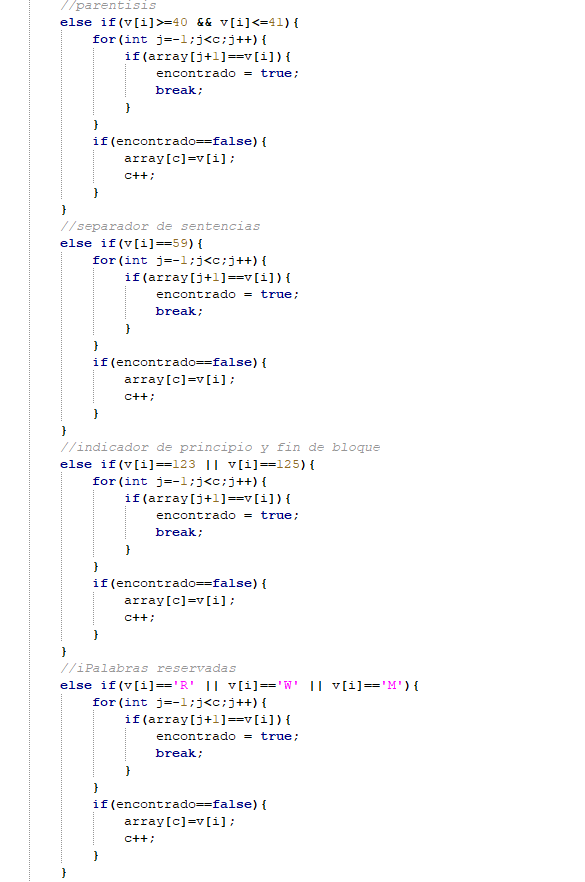
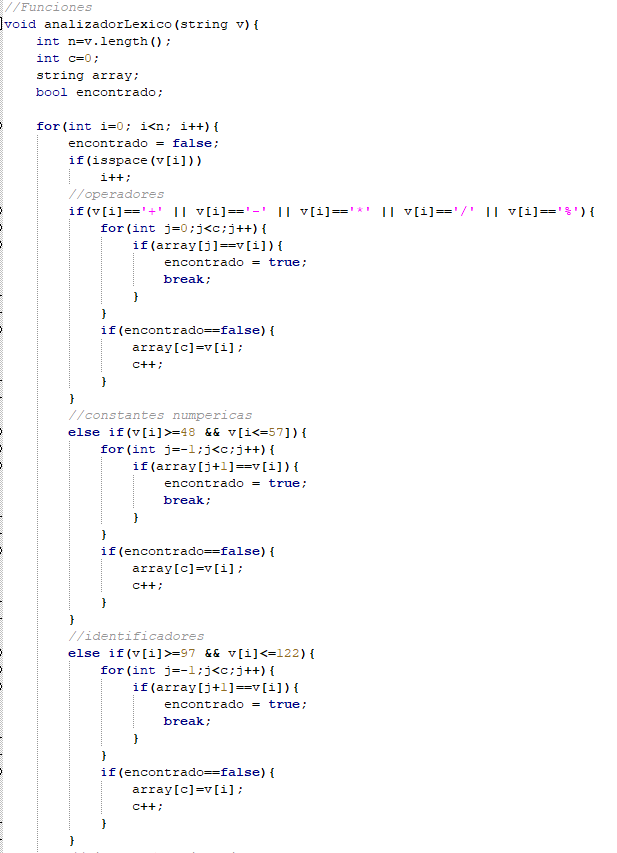
|  |  |
| --- | --- |
| [1] | A. Saénz, «Ampliacion de un compilador para un lenguaje temporal de especificacion de eventos,» Madrid, 2018. |
| [2] | R. Arraeta, Análisis Léxico, Cartagena: Alba, 2013. |
| [3] | R. Vega, Compilador de pseudocódigo como herramienta para el aprendizaje en la construcción de algoritmos, Barranquilla, 2014. |
| [4] | J. Villar, «Analizador Léxico,» Universitat Jaume I, España, 2009. |
| [5] | J. Vásquez, Enseñanzas de la Implementación de un Analizador Léxico, Córdoba, 2015. |
| [6] | G. Alvarez, «Compiladores: Estructura y Procesos de Compilación,» Trujillo, 2014. |
| [7] | J. Ruiz, Compiladores: Teoría e Implementación, R. Libros, Ed., 2010, p. 423. |
| [8] | B. Arcay y C. Dafonte, «Compiladores,» Coruña, 2015. |
| [9] | R. Soto, «Teoría de Autómatas y Compiladores,» Valparaiso , 2010. |
| [10] | N. Barquilla, «Autómatas y Computabilidad,» Madrid, 2019. |
| [11] | C. Hernandez y F. Ortega, «Ciencias de la Compuacion I,» 2009. |
| [12] | «CIDECAME,» 16 Junio 2020. [En línea]. Available: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/unidad\_1\_introduccin.html. |
| [13] | P. Novara, «ZinjaI,» 2019. [En línea]. Available: http://zinjai.sourceforge.net/. |
| [14] | A. Carrillo, Fundamentos de programación en C++, Delta publicaciones, 2005. |
| [15] | E. Jurado, Teoría de autómatas y lenguajes formales, Pedro Cid, S.A.: Cáceres (España), 2008. |
| [16] | J. Cueva, «Análisis Sintáctico en Procesadores de Lenguaje,» Universidad de Oviedo, Oviedo, 1995. |
| [17] | C. S. Infante, «desarrolloweb.com,» 24 Febrero 2006. [En línea]. Available: https://desarrolloweb.com/articulos/2387.php. |
| [18] | E. J. Villegas, «lengyaut.blogspot.com,» 25 Agosto 2017. [En línea]. Available: http://lengyaut.blogspot.com/2017/08/definicion-alfabetos-cadena-lenguaje.html. |
| [19] | U. E. d. Madrid, «www.cartagena99.com,» 12 Abril 2016. [En línea]. Available: http://www.cartagena99.com/recursos/alumnos/apuntes/ININF2\_M4\_U2\_T1.pdf. |
| [20] | S. Hernández, «www.academia.edu,» 07 Agosto 2014. [En línea]. Available: https://www.academia.edu/30969087/Lenguajes\_y\_Aut%C3%B3matas\_1. |
| [21] | Universidad Europea de Madrid, Análisis Léxico, Madrid, 2016. |
| [22] | N. La Serna, Un analizador sintáctico eficiente para las gramáticas del español, 2014. |

1. **ANEXOS**

**Avance del código fuente:**

****

****

****