Analizador Léxico de Pascal
Teoría Computacional

Jesús Omar Anguiano Rosas
Universidad Politécnica de San Luis Potosí
150853@upslp.edu.mx

***Abstract*—The lexical analyzer, this component of a compiler can be used independently to treat files where it is necessary to execute an action based on the recognition of the symbols that interest us. In this document, the system described consists of the error handling tool. Error handling reads tokens and uses automata to validate the string. If an error is detected, this tool sends a message with information about the error.**

I. Introducción

El siguiente documento se redacta con carácter de trabajo académico, esperando además de obtener una competencia asertiva; mostrar un conjunto y desarrollo de implementar un analizador léxico, esperando que sean de gran utilidad por parte de quien lo lea.

El compilador deriva su nombre de la forma en que funciona, analizando el código fuente y recolectando las instrucciones. El proceso de compilación es una secuencia de varias fases. El propósito de este artículo es comprender las tareas realizadas por el analizador léxico y cómo se realizan. Este artículo describe cómo construir una de las fases de un compilador, el léxico. para un lenguaje de programación, en este caso para PASCAL.

II. Marco Teórico

Para que el analizador léxico consiga el objetivo de dividir la entrada en partes, tiene que poder decidir por cada una de esas partes si es un componente separado y, en su caso, de que tipo. De forma natural, surge el concepto de categoría léxica, que es un tipo de símbolo elemental del lenguaje de programación. Por ejemplo: identificadores, palabras clave, números enteros, etc.

A. Antecedentes

Un Analizador Léxico, también se le conoce como Scanner, tiene como función leer el código fuente y crear tokens a partir de la tabla de símbolos, los tokens son grupos de entidades (identificadores, palabras reservadas, separadores, etc).

La función principal es agruparlos en lexemas los caracteres de entrada del código fuente y producir como salida una secuencia de token para cada lexema en el programa fuente. Los tokens obtenidos se mandarán al analizador sintáctico. Para esto se interactúa con la tabla de símbolos.

Es fácil reconocer el patrón que describe a los lexemas que coinciden cuando se leen en la entrada, pero en ciertos lenguajes, no es tan evidente cuando un lexema que corresponde a un token de una instancia. Por ejemplo:

Begin<es la palabra reservada para empezar un programa en pascal>

End.<es la manera de como la palabra reservada end termina una sentencia en pascal>

Se puede investigar ciertas formas para optimizar la importante tarea de leer el programa fuente, pero esta tarea se vuelve difícil ya que a menudo se buscan uno o más caracteres más allá del siguiente lexema para asegurar de que se tiene el lexema correcto, por tanto para poder especificar patrones de lexemas se utilizan expresiones regulares, dado que no se pueden expresar todos los patrones posibles, si se pueden especificar los tipos de patrones que en realidad necesitamos para los tokens.

B. Teoremas en los que se basa el interprete

Realizar un analizador léxico sumamente es una tarea que requiere de cierto tipo de programación, es decir, no cualquier lenguaje puede compilar un intérprete, pues el uso de tokens necesitan el uso de una memoria o un buffer que será almacenado en el ordenador, cuando se entiende esto, se puede tratar el problema utilizando bancos de memoria como pilas o bien completar autómatas finitos muy largos que hagan la tarea recursivamente, me he seguido por ciertos diagramas de transición vistos en clase y que a continuación expongo.

Los diagramas de transición de estado son diagramas de flujo utilizados que se convierten de los patrones, cuenta con estados que son conexiones de nodos, cada estado representa una condición que podría ocurrir durante el proceso de explorar la entrada, buscando un lexema que coincida con uno de varios patrones.

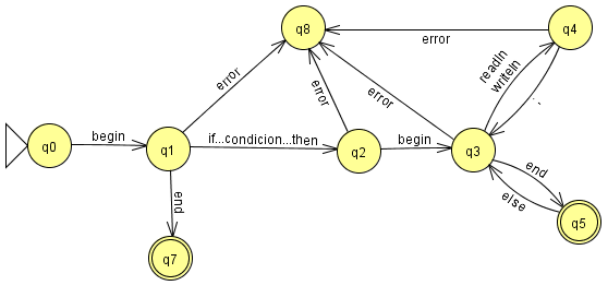
Las líneas se dirigen de un estado a otro del diagrama de transición de estados, cada línea se etiqueta mediante un símbolo o conjunto de símbolos.

Si nos encontramos en cierto estado s, y el siguiente símbolo de entrada es a, buscamos una línea que salga del estado s y esté etiquetado por a (y tal vez por otros símbolos

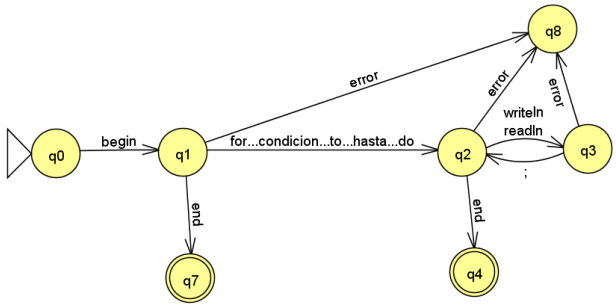
también). Si encontramos dicha línea, avanzamos el apuntador avance y entramos al estado del diagrama de transición de estados al que nos lleva esa línea.

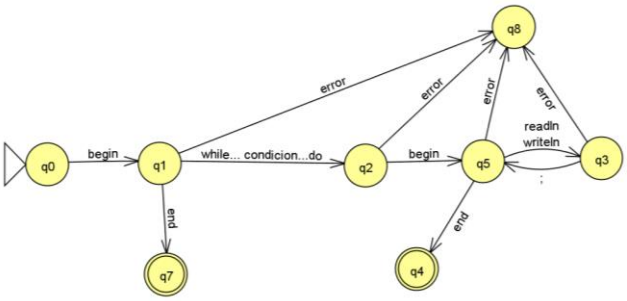
C. Desglose de los autómatas utilizados

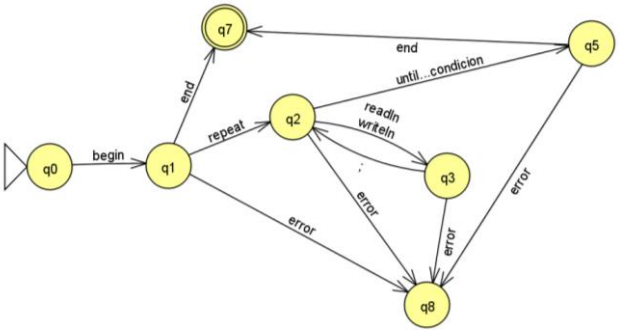
En el corazón de la transición se encuentra el formalismo conocido como autómatas finitos. Para cada estado q y cada transición por palabras, obteniendo una visión general y símbolo de entrada a. representativa del funcionamiento.
Empezamos con las instrucciones del programa, comenzando por la sentencia por los condicionales en este caso el If:



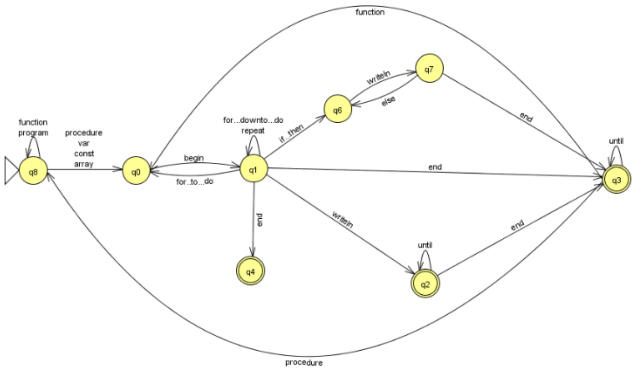
Enseguida continuamos con los bucles que son el for y el while







Un desglose mas en general de como se desarrollo el programa mostramos un autómata el cual nos representa todo un proceso en el lenguaje juntando cada uno de los autómatas de los ciclos.



Sin duda este último autómata sirvió para tomar las base y llevar a cabo la elaboración del Código de programación, elaborado en KOTLIN.

D. Secuencia de los pasos para el desarrollo y la solución del problema

Para la creación del programa sobre el analizador léxico de pascal, se presentaron algunos puntos por analizar con la estructura de los autómatas ya que al realizarlos por serados dependiendo la función de cada uno de las palabras reservadas, al hacer uno solo en general fue más difícil de lo esperado.

E. Resultados y conclusiones

Los resultados obtenidos son poco satisfactorios, si bien realizar un analizador léxico conlleva un arduo trabajo, un uso de tokens complejos y una programación más avanzada, el proyecto sirvió como una perfecta guía para conocer los conceptos o principios básicos del funcionamiento y comprender como un autómata, conlleva a un gran algoritmo en un compilador de código.

Este proyecto nos deja un gran conocimiento ya que si realmente no se realizó un compilador como tal, si se buscó realizar uno de los pilares para que un compilador funcione de la mejor manera si como es el analizador léxico.

Glosario—

1. **Alfabeto:** Alfabeto es un conjunto finito y no vacío de símbolos, letras o caracteres como, por ejemplo: las letras del abecedario, los dígitos del sistema decimal, los operadores aritméticos, etc.
2. **Autómata:** Un autómata es una máquina procesadora de información que recibe un conjunto de señales de entrada, y produce en correspondencia un conjunto de señales de salida.
3. **Estado:** es el resumen de lo acontecido (entradas previas) hasta un determinado instante por lo que la máquina puede recordar "que ha sucedido en el pasado". Una máquina puede tener un cierto número de estados correspondientes a un cierto número de clases distintas de historias pasadas. Una máquina con un número finito de estados se llama maquina o autómata de estados finito
4. **Expresiones regulares:** Es una notación que permite definir de manera precisa un lenguaje sobre un alfabeto. Una expresión regular se construye a partir
   de expresiones regulares más simples utilizando un conjunto de reglas definitorias.
5. **Gramática:** Una gramática es un ente formal para especificar, de una manera finita, un conjunto de sentencias, o cadenas de símbolos, potencialmente infinito (y que constituye un lenguaje). Las cadenas o palabras de un lenguaje son generadas por la gramática, empezando por una cadena que consiste en un símbolo particular denominado símbolo inicial y rescribiendo sucesivamente la cadena, de acuerdo con un conjunto finito de reglas o producciones.
6. **Lenguajes:** Desde el punto de vista lingüístico, un lenguaje es un conjunto finito o infinito de oraciones, cada una de ellas de longitudes finitas y construidas por concatenación a partir de un número finito de elementos. Desde el punto de vista informático, un lenguaje es una notación formal para describir algoritmos o funciones que serán ejecutadas por una computadora. No obstante, la primera definición es válida para ambos puntos de vista