Universidad Rafael Landívar

Facultad de ingeniería

Ingeniería en Informática y Sistemas

Lenguajes formales y autómatas

Catedrático: Mgtr. Moisés Alonso

# DISEÑO DE APLICACIÓN – ANALIZADOR LÉXICO

Nombre y carné:

Marcos Andrés Calderón Morales (1060918)

Guatemala, 04 de marzo del 2020

# Introducción

La programación es el proceso para solucionar problemas a través de la definición de un programa. Esta cuenta con paradigmas, que representan enfoques diferentes para la construcción de soluciones a problemas. También cuenta con metodologías, las cuales son todas las técnicas y conocimientos necesarios para el funcionamiento de un programa informático.

Este proyecto consistió en desarrollar la primera fase de un compilador, consistente en un programa que recibe como entrada un archivo de texto y produce una salida compuesta de tokens (componentes léxicos) o símbolos.

En base a ello, se ha desarrollado una simulación del analizador, en el que cada archivo que analiza está formado de:

1. SETS
   1. La palabra “SETS” debe estar en mayúscula.
   2. Pueden estar concatenados a través del signo “+”.
   3. Se puede utilizar la función CHR
   4. Contiene la definición abreviada de un conjunto de símbolos terminales.
   5. Puede haber muchos espacios en blanco entre el identificador, el símbolo “=” y la definición
   6. Esta parte puede o no venir.
      1. si aparece, debe poseer al menos un SET.
2. TOKENS
   1. Representan los símbolos terminales y no terminales de la gramática.
   2. La palabra TOKENS debe existir y estar en mayúscula.
   3. Esta sección debe existir.
   4. Cada token debe poseer la palabra: TOKEN y un número, seguido del signo igual “=”.
3. ACTIONS
   1. Contiene definición de funciones, en este caso específico las palabras reservadas del lenguaje.
   2. Reservadas() siempre debe existir.
   3. Puede haber otras funciones.
4. ERROR
   1. Debe venir al menos uno,
   2. ERROR debe tener asignado un número.
   3. El identificador debe tener como sufijo la palabra ERROR en mayúscula

Si alguna de estas características no las trae consigo el archivo de texto, el resultado deberá ser indicándole al usuario que existe un error y la posición del primer error.

De lo contrario, salida será un indicador mostrando que el archivo no contiene ningún error de gramática.

# OBJETIVOS

## General

Desarrollar una simulación del analizador léxico, la primera fase para la creación de un compilador.

## ESPECIFICOS

1. Crear expresiones regulares que permitan aceptar un conjunto de caracteres.
2. Crear árboles de expresión para poder evaluar un conjunto de caracteres.
3. Mostrar si el archivo tiene error, e indicar la posición en la que se encuentra.
4. Mostrar si el archivo de entrada no tiene error.

# Expresiones regulares

SETS

Probada en Regex101: \*([A-Za-z])+ \*= \*('[A-Za-z]'|CHR\([0-9]+\))((..|\+)('[A-Za-z]'|CHR\([0-9]+\)))\*

En el programa: ( \*.L+. \*.=. \*.(('.S.')|(C.H.R.\\(.N+.\\))).(((\\..\\.)|\\+).(('.S.')|(C.H.R.\\(.N+.\\))). \*)\*)

TOKENS

Probada en Regex101: \*TOKEN +[0-9]+ \*= \*(('?.'?|[A-Z]+) \*)+

En el programa: ( \*.T.O.K.E.N.Z\*.N+.Z\*.=.Z\*.((('.S.')|S+). \*)+)

ACTIONS

Probada en Regex101: \*TOKEN +[0-9]+ \*= \*(('?.'?|[A-Z]+) \*)+

En el programa: ( \*.N+. \*.=. \*.'.L+.')

ERROR

Probada en Regex101: \*TOKEN +[0-9]+ \*= \*(('?.'?|[A-Z]+) \*)+

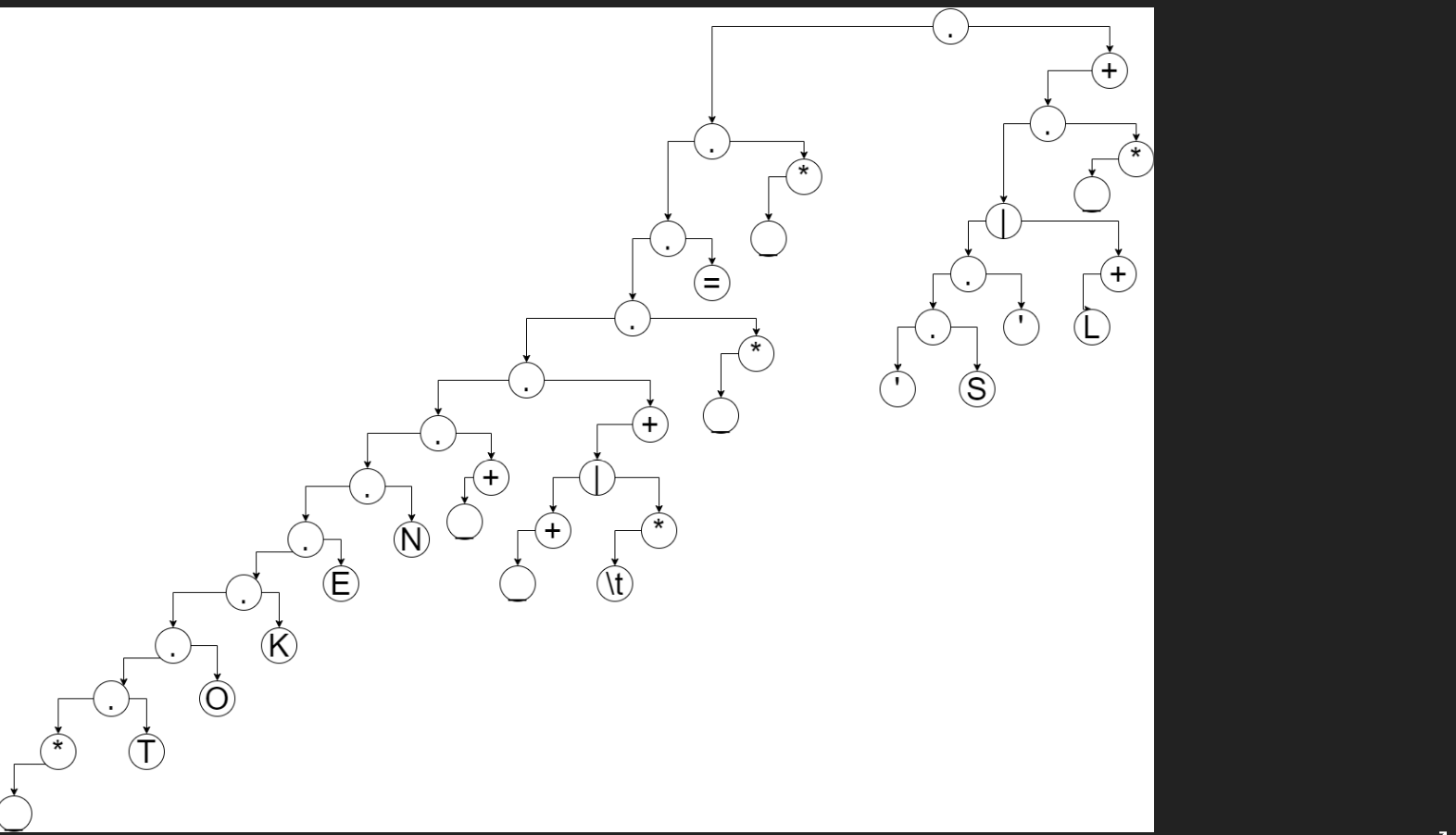
En el programa: ( \*.E.R.R.O.R. \*.=. \*.N+)

# Árboles de expresión

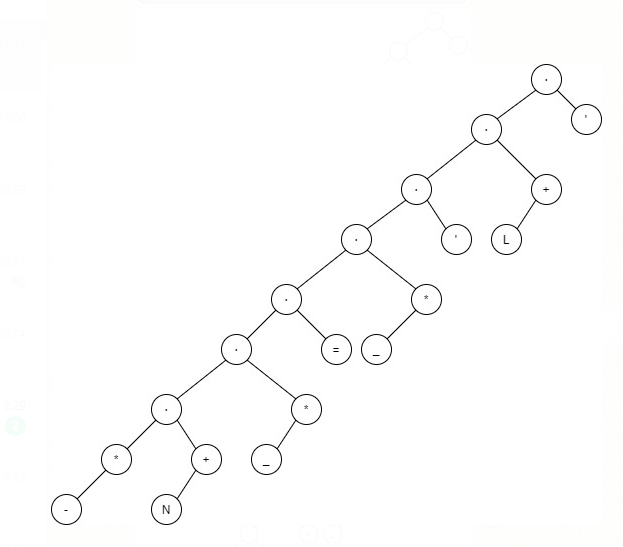
SETS

# 

TOKENS



ACTIONS



ERROR

