

**Analizador Léxico**

**Integrantes**

Farfán de León José Osvaldo

Sección: D06

Salcedo Arellano Alexa

Sección: D06

Víctor Manuel Pacheco Moreno

Sección: D03

**Materia**: Traductores de Lenguajes ll

**Profesor**: Ramos Barajas Armando

Índice

[Introducción 3](#_Toc95499759)

[¿Qué es un analizador léxico? 3](#_Toc95499760)

[Características 3](#_Toc95499761)

[Función 3](#_Toc95499762)

[**Objetivos:** 3](#_Toc95499763)

[General 4](#_Toc95499764)

[Particular 4](#_Toc95499765)

[Desarrollo: 4](#_Toc95499766)

[Apéndices 7](#_Toc95499767)

[Acrónimo 10](#_Toc95499768)

[Diagramas 10](#_Toc95499769)

[Requisitos funcionales 10](#_Toc95499770)

[Requisitos no funcionales 10](#_Toc95499771)

[Complejidad ciclomática 11](#_Toc95499772)

[COCOMO 12](#_Toc95499773)

[Caja negra 12](#_Toc95499774)

[Caja blanca 14](#_Toc95499775)

[Grafos 17](#_Toc95499776)

[Tabla de transiciones 17](#_Toc95499777)

[Conclusión: 18](#_Toc95499778)

[Bibliografía 18](#_Toc95499779)

# Introducción

En esta actividad se buscará la implementación de un analizador léxico en un lenguaje de programación, del cual seleccionamos c/c++, debido a que es el lenguaje con el cual tenemos un mayor conocimiento, en esta parte de nuestro compilador buscaremos crear el primer parte del mismo la cual consiste en el análisis léxico de los datos que sean introducidos por el usuario, definir la palabra introducida por el usuario e identificar su tipo.

## ¿Qué es un analizador léxico?

Un analizador léxico se encarga de buscar los componentes o palabras que componen el programa fuente, según unas reglas o patrones. La entrada del analizador léxico podemos definirla como una secuencia de caracteres.

## Características

El analizador léxico tiene que dividir la secuencia de caracteres en palabras con significado propio y después convertirlo a una secuencia de terminales desde el punto de vista del analizador sintáctico, ya que es su entrada. El analizador léxico reconoce las palabras en función de una gramática regular de manera que sus no terminales se convierten en los elementos de entrada de fases posteriores. En Lex, por ejemplo, esta gramática se expresa mediante expresiones regulares.

## Función

El analizador léxico es la primera fase de un compilador. Su principal función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis. Esta interacción, suele aplicarse convirtiendo al analizador léxico en una subrutina o corrutina del analizador sintáctico. Recibida la orden “Dame el siguiente componente léxico” del analizador sintáctico, el analizador léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico.

# **Objetivos:**

## General

Aprender a implementar los algoritmos y autómatas aprendidos en la materia de teoría de la computación. Así como el tener un claro el correcto funcionamiento de cada una de las fases de un compilador. Tener la capacidad de generalizar los algoritmos implementados para resolver problemas de traducción de lenguajes, procesamiento de lenguajes y reconocimiento de patrones.

Su principal función es la de ser un analizador léxico el cual sea capaz de leer el flujo de caracteres de entrada y transformarlo a una secuencia de componentes léxicos capaz de ser empleada por el analizador sintáctico. Al tiempo que realiza esta función, el analizador léxico se ocupa de ciertas labores de “Limpieza”

## Particular

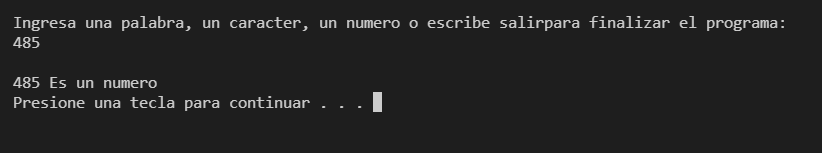
Aprender el correcto uso de los autómatas finitos deterministas en la aplicación de los analizadores léxicos y sus símbolos en la creación de un lenguaje de programación de alto nivel.

Simular un compilador, tomando en cuenta únicamente el analizador léxico

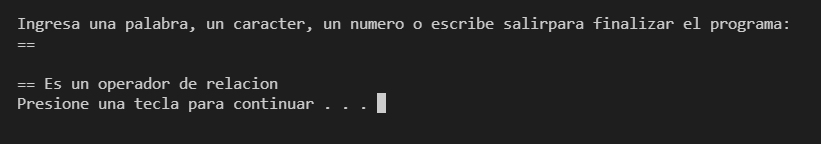
Interpretar la forma en que un compilador realiza el análisis léxico de un programa.

# Desarrollo:

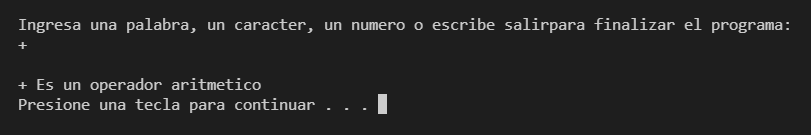
El analizador léxico puede identificar números enteros:

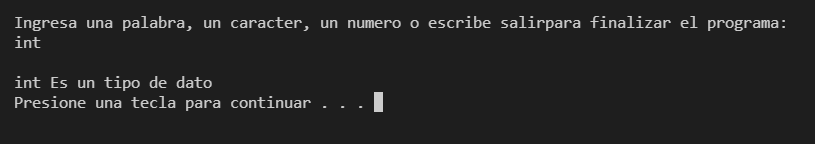


El analizador léxico puede identificar operadores relacionales ( <, <=, >, >=, ==):

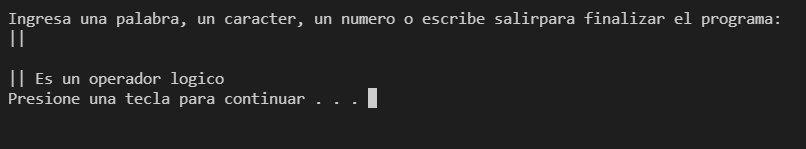


El analizador léxico puede identificar operadores aritmeticos (+, -, /, \*, pow, %, sqrt)

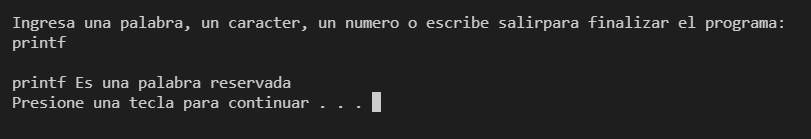


El analizador léxico puede identificar tipos de datos (int, float, char, string, double, bool)

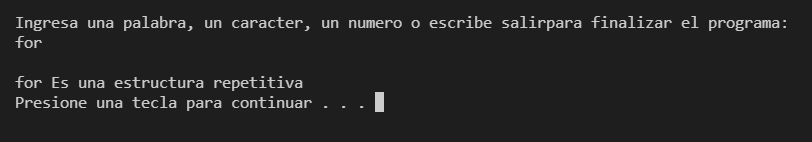
El analizador léxico puede identificar operadores logicos(||, &&)



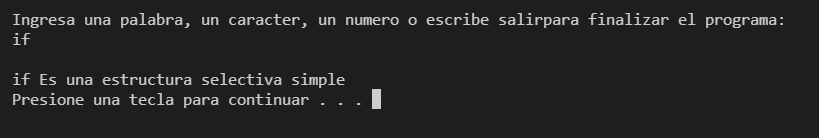
El analizador léxico puede identificar palabras reservadas(printf, scanf, gets, puts, case, break, return)



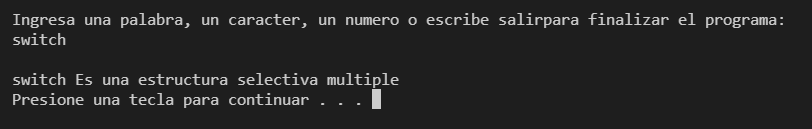
El analizador léxico puede identificar estrucuras repetitivas(for, while, do)



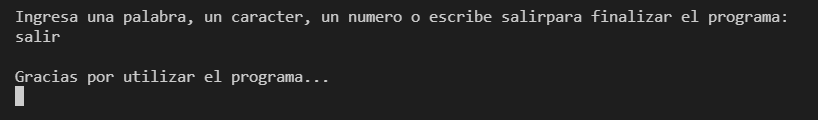
El analizador léxico puede identificar estructuras selectivas(if, else)



El analizador léxico puede identificar estructuras selectivas múltiples(switch)



El analizador léxico puede identificar la salida.



# Apéndices

#include <iostream>

#include <string.h>

#include <cstring>

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

using namespace std;

bool isNumber(char txt[]){

bool ban = false;

for(int i=0; i < strlen(txt); i++){

if(!isdigit(txt[i]) && txt[i] != '-'){

if (txt[i] != '.'){

return false;

}

else{

ban = true;

}

}

}

if(ban == true){

cout << "\n" << txt << " Es un flotante" << endl;

system("pause");

return true;

}

cout << "\n" << txt << " Es un entero" << endl;

system("pause");

return true;

}

bool isString(char txt[]){

for (int i=0; i< strlen(txt); i++){

char actual = txt[i];

if(actual == '"'){

cout << "\n" << txt << " Es una cadena" << endl;

system("pause");

return true;

}

}

cout << "\n" << txt << " Es un identificador" << endl;

system("pause");

return false;

}

int main(){

char texto[15];

bool valido = true;

while(valido == true){

system("cls");

cout << "Ingresa una cadena: "<<endl;

cin >> texto;

if(isNumber(texto)){

valido = true;

continue;

}

if (strcmp(texto, "<") == 0 || strcmp(texto,"<=") == 0 || strcmp(texto,">") ==0 || strcmp(texto,">=") ==0 || strcmp(texto,"==") == 0 || strcmp(texto,"!=") == 0){

cout << "\n" << texto << " Operador de relacion" << endl;

system("pause");

valido = true;

continue;

}

if (strcmp(texto, "+") == 0 || strcmp(texto,"-") == 0 || strcmp(texto,"\*") ==0 || strcmp(texto,"/") ==0 || strcmp(texto,"%") == 0 || strcmp(texto,"sqrt") == 0){

cout << "\n" << texto << " Operador aritmetico" << endl;

system("pause");

valido = true;

continue;

}

if (strcmp(texto, "int") == 0 || strcmp(texto,"char") == 0 || strcmp(texto,"float") ==0 || strcmp(texto,"double") ==0 || strcmp(texto,"string") == 0 || strcmp(texto,"bool") == 0){

cout << "\n" << texto << " Tipo de dato" << endl;

system("pause");

valido = true;

continue;

}

if (strcmp(texto, "||") == 0 || strcmp(texto,"and") == 0 || strcmp(texto,"!") ==0){

cout << "\n" << texto << " Operador logico" << endl;

system("pause");

valido = true;

continue;

}

if (strcmp(texto, "print") == 0 || strcmp(texto,"input") == 0 || strcmp(texto,"get") ==0 || strcmp(texto,"put") ==0 || strcmp(texto,"break") == 0 || strcmp(texto,"return") == 0){

cout << "\n" << texto << " Palabra reservada" << endl;

system("pause");

valido = true;

continue;

}

if (strcmp(texto, "for") == 0 || strcmp(texto,"while") == 0 || strcmp(texto,"do") ==0){

cout << "\n" << texto << " Ciclos" << endl;

system("pause");

valido = true;

continue;

}

if (strcmp(texto, "if") == 0 || strcmp(texto,"elif") == 0 || strcmp(texto,"else") ==0){

cout << "\n" << texto << " Condiciones" << endl;

system("pause");

valido = true;

continue;

}

if (strcmp(texto, "switch") == 0 ){

cout << "\n" << texto << " Estructura selectiva" << endl;

system("pause");

valido = true;

continue;

}

if (strcmp(texto, "salir") == 0 ){

cout << "\n" << texto << " Saliendo ..." << endl;

system("pause");

valido = false;

continue;

}

else{

isString(texto);

continue;

}

}

return 0;

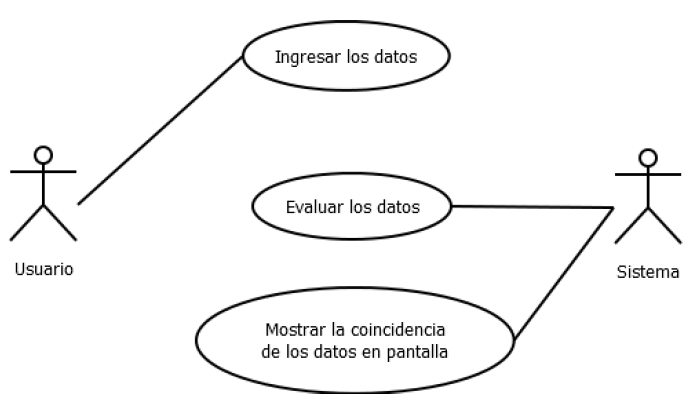
}

# Acrónimo

Cocomo -> COnstructive COst MOdel

# Diagramas

caso de uso



# Requisitos funcionales

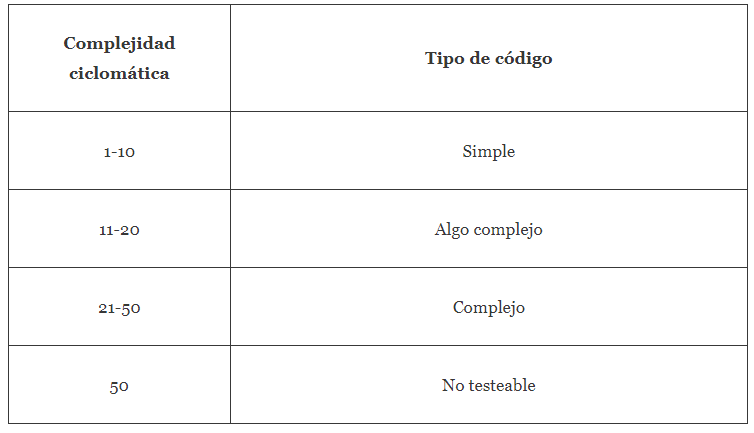
* El diagrama debe leer la entrada de los datos de usuario
* El programa validará si dichos datos se encuentran en las opciones que se le asignaron.
* En caso de que, si se encuentre, se mostrará información relacionada a dichos datos, de lo contrario simplemente se solicitará una nueva entrada de datos

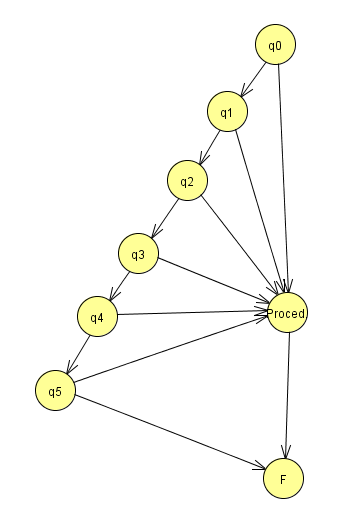
# Requisitos no funcionales

* Limpieza de pantalla automaticamente

# Complejidad ciclomática

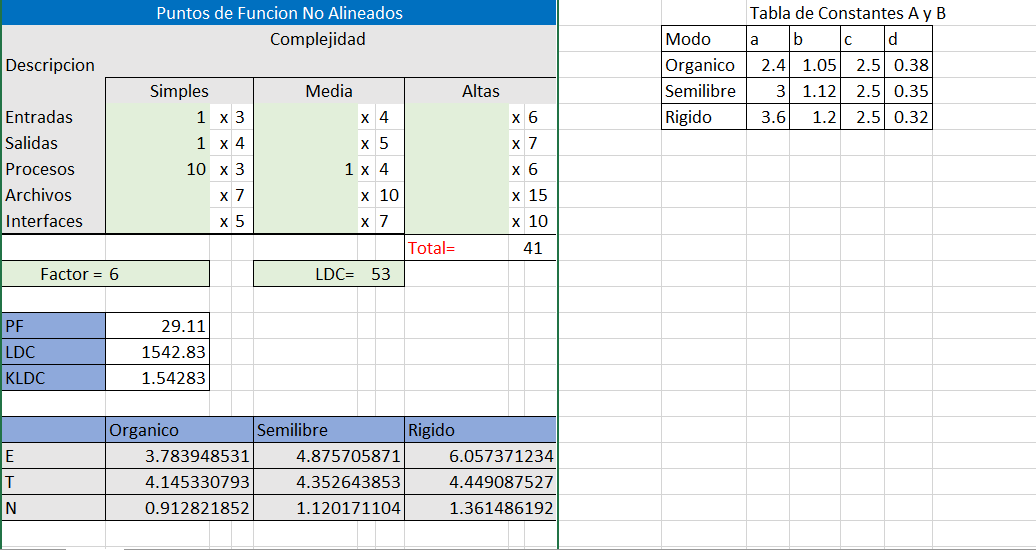
* Restar las **aristas** menos los **nodos** y sumar 2:
  + V(G) = Aristas – Nodos + 2
  + V(G) = 42 – 34 + 2 = 10



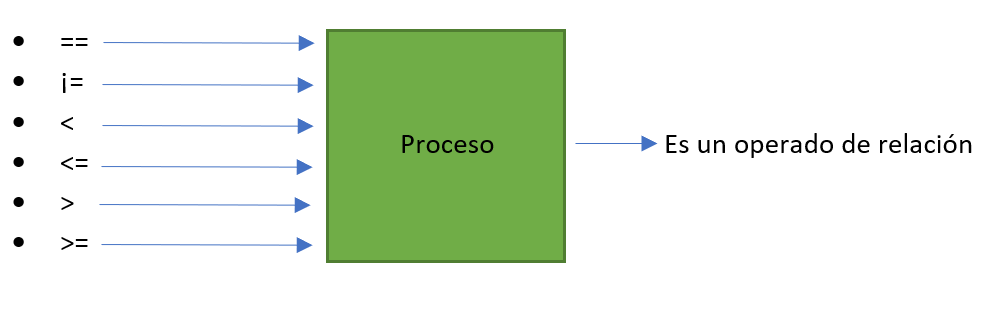


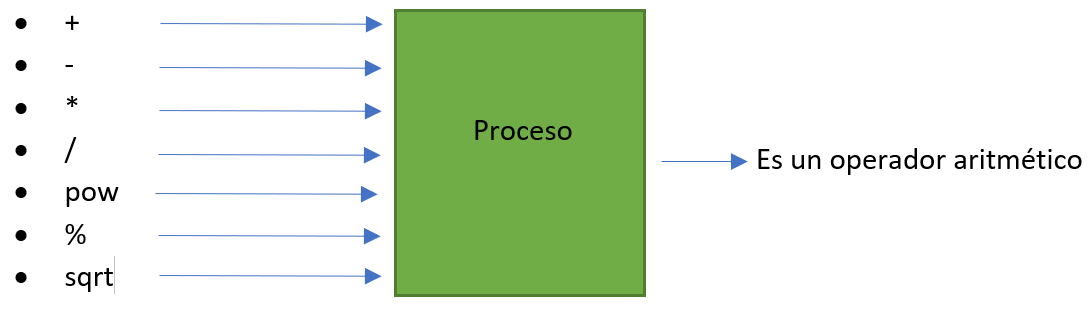
* + V(G) = 13 – 8 + 2 = 7

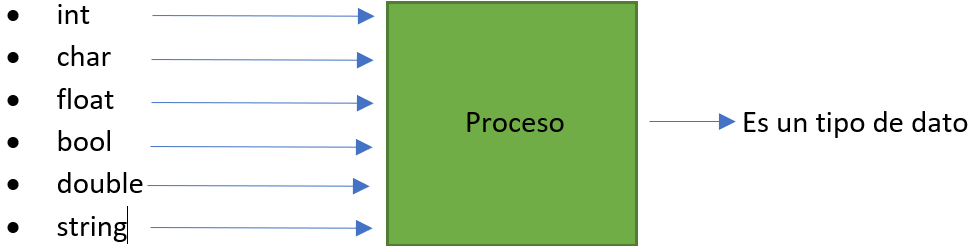
# COCOMO

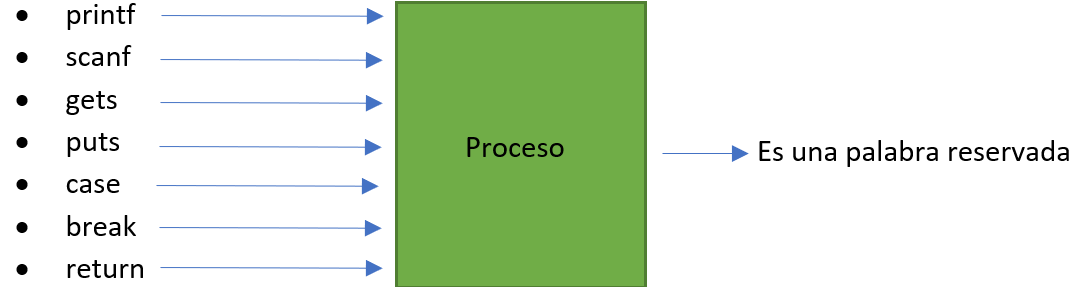


# Caja negra







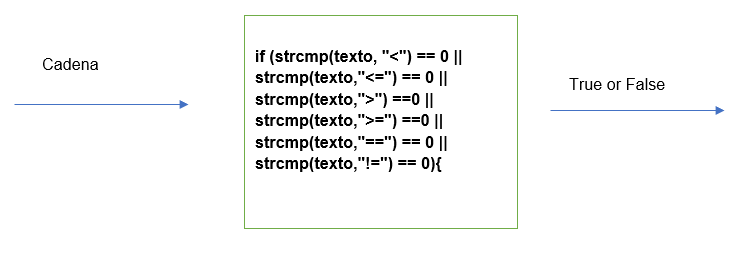


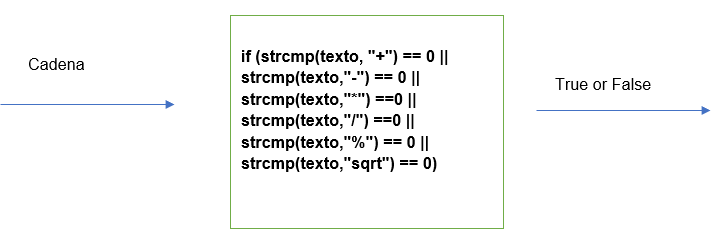


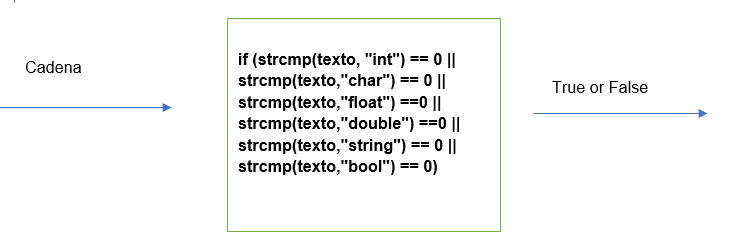


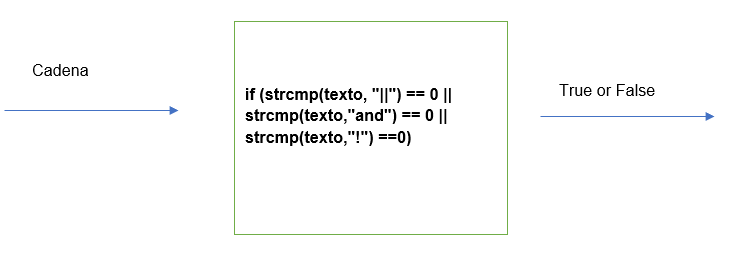


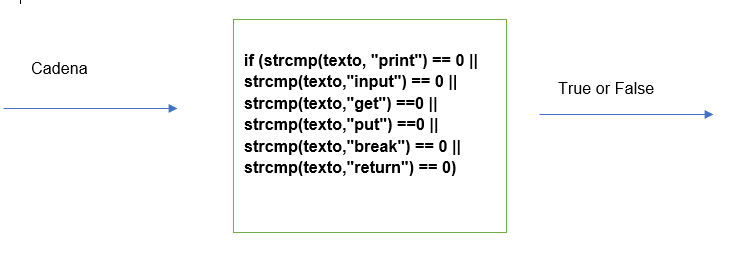
# Caja blanca

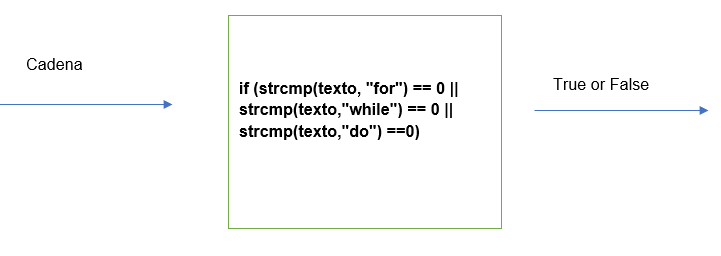
****

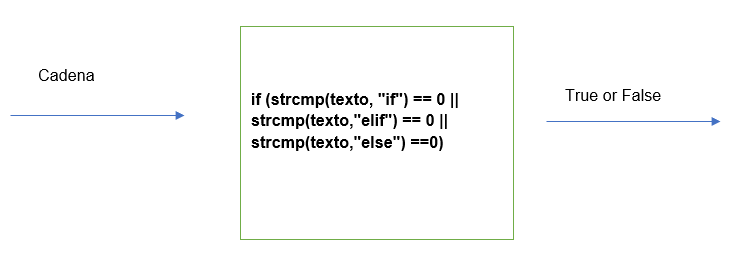
****

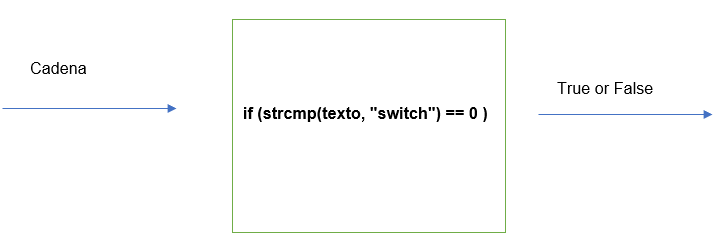
****

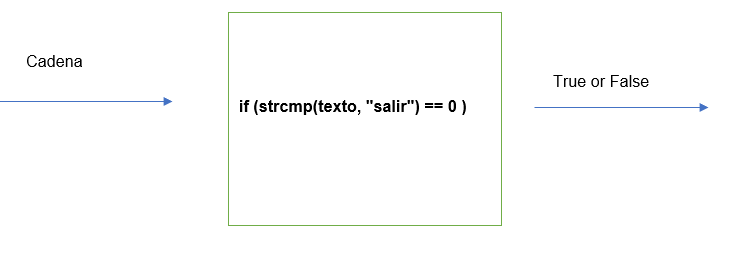
****

****

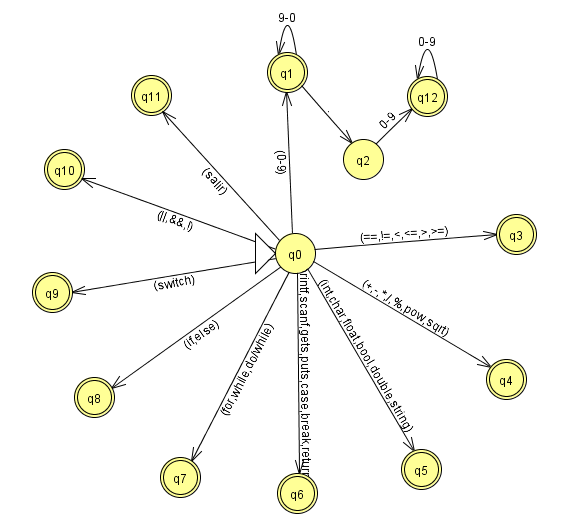
****

****

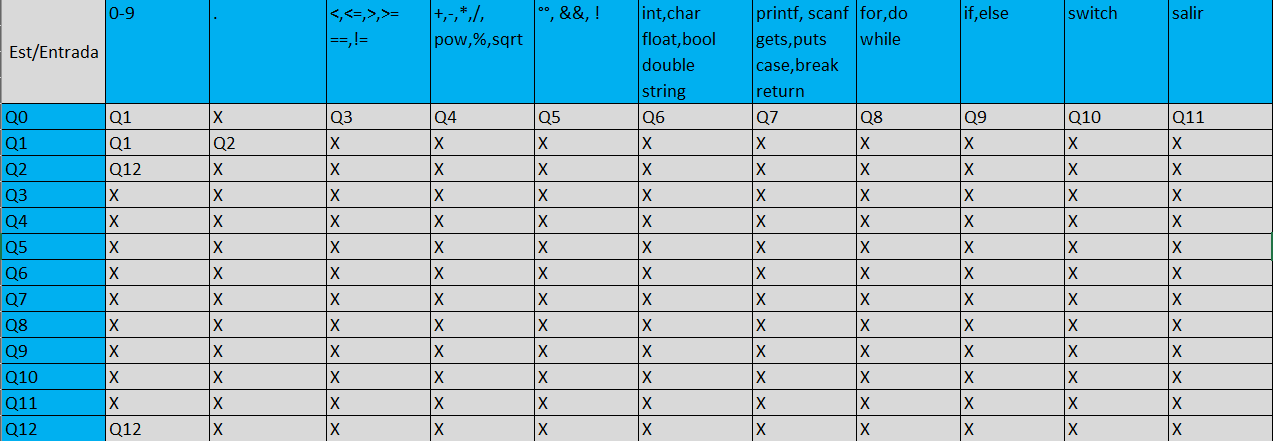
****

****

# Grafos



# Tabla de transiciones



# Conclusión:

Con esta práctica pudimos darnos cuenta de la vital importancia de lo que es un analizador léxico ya que es el principio de un lenguaje de programación que se encarga de buscar los componentes o palabras que componen el programa fuente, según las reglas o patrones definidos en dicho lenguaje, siempre obteniendo una secuencia de caracteres la cual podemos analizar y clasificar como x tipo, también en el proceso nos dimos cuenta de lo que seria la complejidad ciclomatica que define que tan difícil es el código simplemente realizando lo que es un grafo sencillo en base a la composición del grafo con los if else, switch o ciclos es muy sencillo de sacarlo.

# Bibliografía

Gálvez, S.. (2020). Tema 2. Análisis lexicográfico. Traductores,

Compiladores e Intérpretes(pp.1-15). Jalisco, México : UMA.

Aho, A.V., Sethi, R., Ullman, J.D. (1990), Compiladores: principios, técnicas y herramientas, Tema 3, p´aginas: 85-158.

Louden, K.C. (1997), Compiler Construction: Principles and Practice, Tema 2, paginas: 31-93.