
Implementación de métodos computacionales (Gpo 2)

Programando un DFA

Analizador Léxico

# Manual de usuario

Hecho por:

Lizbeth Maribel Melendez Delgado A01232559

Gerardo Novelo De Anda A01638691

26/Marzo/2021

# Contenido

# 1. Introducción

## 1.1 Objetivo

¡Hola!, el siguiente documento presenta una breve descripción y serie de pasos que te recomendamos seguir para poder correr el programa Analizador Léxico, el cual es una implementación en el lenguaje Java de un autómata finito determinista para reconocer tokens en un lenguaje de programación.

## 1.2 Requerimientos

Para correr el programa necesitarás contar con:

- Archivos del programa
- IDE instalado (recomendamos Visual Studio Code)
- Lenguaje Java

Más adelante en el apartado 3.1 Instalación de programas, proporcionamos algunos links que te serán de ayuda para la descarga y el proceso de instalación de cada uno de estos requerimientos en caso de no contar con ellos.

# 2. Diseño del programa

## 2.1 Especificaciones del programa

El programa recibe como parámetros de entrada y salida lo siguiente:

- Entrada:

  Un archivo de texto (.txt) que contiene expresiones aritméticas y comentarios, una por renglón, donde cada uno de los token no necesariamente están separados por un espacio en blanco, e incluso puede haber más de un espacio en blanco entre cada uno.

- Salida:

  Una tabla con los distintos tokens encontrados y su tipo.

Las expresiones que se encuentren en el archivo de texto podrán contener los siguientes tipos de tokens con las características que se describen a continuación:

| Descripción | | |
|---|---|---|
| **Token** | **Descripción** | **Ejemplo** |
| Variable | Empieza con una letra (may o min) y contiene letras, números y underscore (‘_’). | a, b, a_1, ab_1c |
| Entero | Sin punto decimal | 7, -1 |
| Real | Positivos o negativos, pueden tener o no parte decimal pero deben contener punto, pueden usar notación exponencial | 32.4, -8.6, 6.1E-8, 2.3E3, 6.345e-5, -0.001E-3, 0.467E9 |
| Asignación | Operadores | = |
| Suma | | + |
| Resta | | - |
| Multiplicación | | * |
| División | | / |
| Potencia | | ^ |
| Paréntesis que cierra | Símbolos especiales | ( |
| Paréntesis que abre | | ) |
| Comentarios | Inician con // y todo lo que sigue hasta que termina el renglón es un comentario | // esto es un comentario 1 + 2 == |

*Tabla 1. Tabla descriptiva*

## 2.2 Autómata finito determinístico

Para la implementación del programa, realizamos un diagrama de autómata finito determinístico, que representa la transición de un estado a otro con base a la entrada de un símbolo.

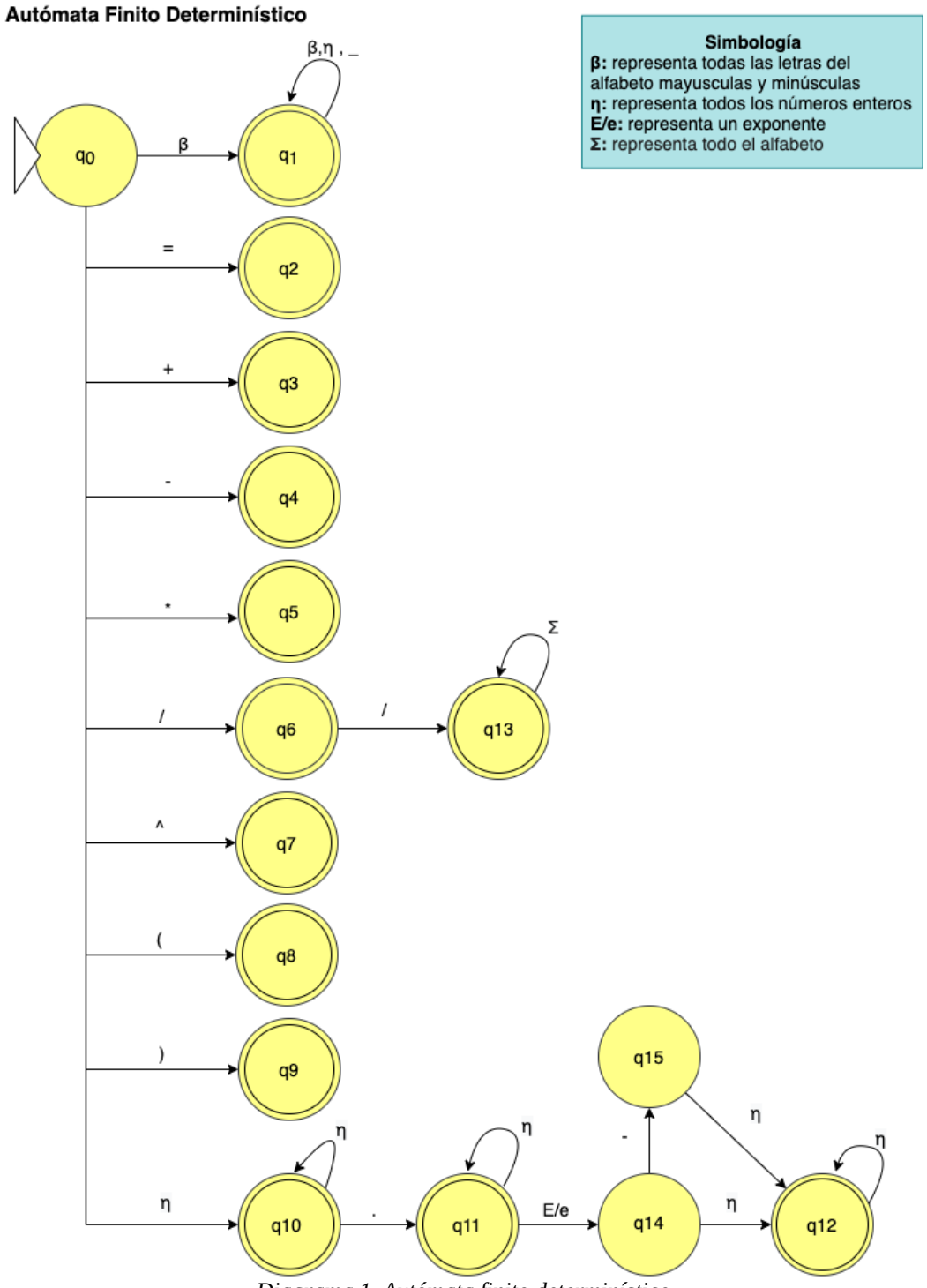



*Diagrama 1. Autómata finito determinístico*

Utilizando nuestro diagrama (*diagrama 1*) representamos su información ahora en una tabla de transición.

| Simbología | |
|---|---|
| ***:** | El estado (1er columna) con * representa el estado inicial |
| **Sombreado:** | Los estados (1er columna) con sombreado verde representan los estados finales |
| **β:** | representa todas las letras del alfabeto mayúsculas y minúsculas |
| **η:** | representa todos los números enteros |
| **E/e:** | representa un exponente |
| **Σ:** | representa todo el alfabeto |

*Tabla 2. Simbología*

| Tabla de transición | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| | **β** | **η** | **_** | **=** | **+** | **-** | ***** | **/** | **^** | **(** | **)** | **E/e** | **.** |
| **q0*** | q1 | q10 | - | q2 | q3 | q4 | q5 | q6 | q7 | q8 | q9 | q1 | - |
| **q1** | q1 | q1 | q1 | - | - | - | - | - | - | - | - | q1 | - |
| **q2** | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **q3** | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **q4** | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **q5** | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **q6** | - | - | - | - | - | - | - | q13 | - | - | - | - | - |
| **q7** | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **q8** | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **q9** | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **q10** | - | q10 | - | - | - | - | - | - | - | - | - | | q11 |
| **q11** | - | q11 | - | - | - | - | - | - | - | - | - | q14 | - |
| **q12** | - | q12 | - | - | - | - | - | - | - | - | - | - | - |
| **q13** | q13 | q13 | q13 | q13 | q13 | q13 | q13 | q13 | q13 | q13 | q13 | q13 | q13 |
| **q14** | - | q12 | - | - | - | q15 | - | - | - | - | - | - | - |
| **q15** | - | q12 | - | - | - | - | - | - | - | - | - | - | - |

*Tabla 3. Tabla de transición*

# 3. Uso del programa

## 3.1 Instalación de programas

*Descargar archivos*

Los archivos del programa los puedes encontrar en el siguiente enlace:

- Programa Analizador en Github
- O ejecuta la siguiente línea de comando en tu terminal

```
git clone https://github.com/marmelendez/archivos_analizador.git
```

Asegurate de que la carpeta contenga el archivo de texto llamado expresiones.txt y un paquete (carpeta) con 3 archivos: Analizador.java, Token.java y Tipo.java

*Descargar e instalar Visual Studio Code*

A continuación te proporcionamos algunos links para descargar e instalar el IDE Visual Studio Code

Para descargar:

- Download Visual Studio Code
- Getting started

Para instalar:

- Setting up Visual Studio Code

*Descargar e instalar Java*

Para descargar y configurar Java en Visual Studio Code ingresa al siguiente enlace:

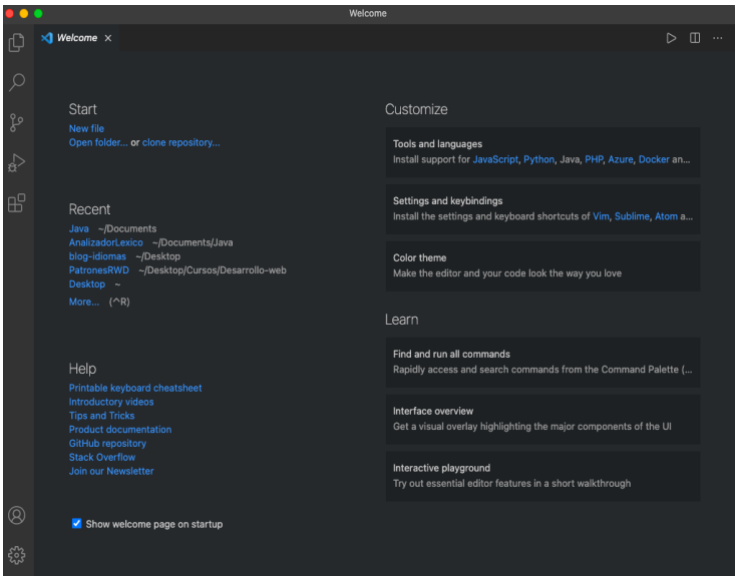
- Getting Started with Java in VS Code

**Nota**: si utilizas otro IDE, solo asegúrate de contar con el lenguaje de programación Java. Cabe mencionar que las instrucciones de como correr el programa se muestran en el siguiente apartado utilizando Visual Studio Code.

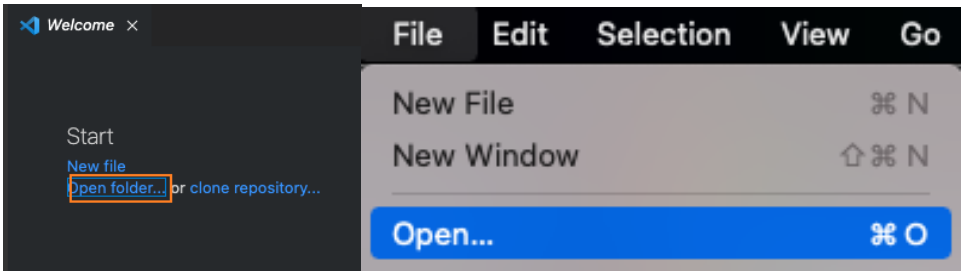
## 3.2 Correr programa

Después de descargar e instalar todos los programas necesarios, ahora si, para poder correr el programa:

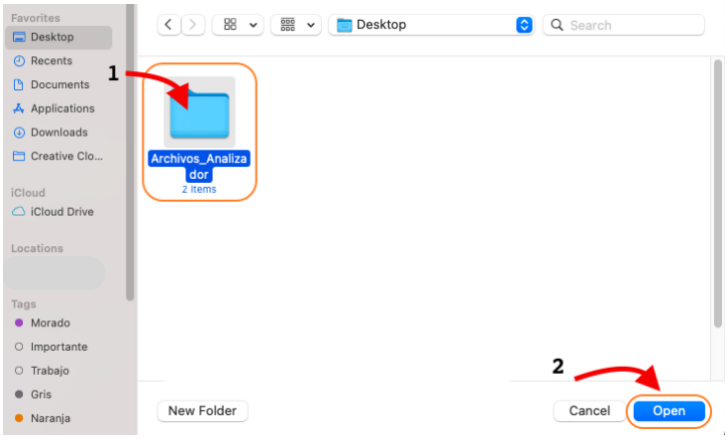
1. Abre Visual Studio Code



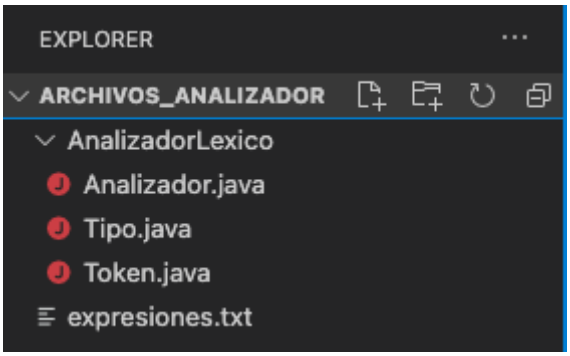

2. Abre una nueva carpeta desde la ventana de inicio de VSC o desde el menú



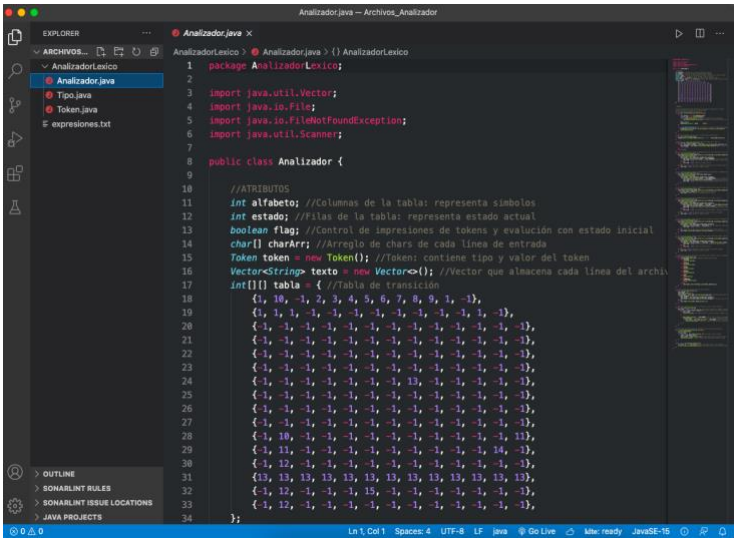

3. Selecciona la carpeta Archivos_Analizador que contiene los archivos que descargaste del programa y ábrela



4. En tu explorador te debe aparecer lo siguiente



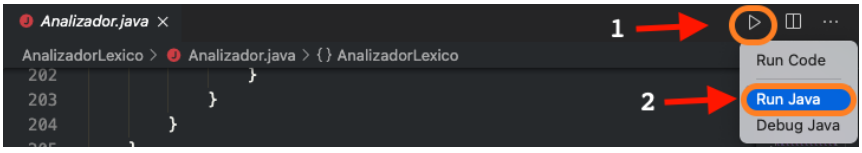

5. Selecciona el archivo Analizador.java



6. Dentro de este archivo se encuentra el main de nuestro programa. Aquí es donde llamamos a la función que se encarga de hacer prácticamente todo, *lexerArtimetico*. Pasamos como parámetro el nombre del archivo.txt entre comillas

```
    public static void main(String[] args) {
        Analizador a = new Analizador(); //Se
        a.lexerAritmetico("expresiones.txt");
    }
}
```

7. Y ahora corremos nuestro programa al dar clic en la flecha arriba del archivo.



## 3.3 Pruebas y resultados esperados

Probablemente te preguntes ¿cómo sé que está funcionando correctamente el programa?, no te preocupes, a continuación enlistamos algunos casos de pruebas con sus resultados para que puedas comprobar el correcto funcionamiento del programa.

Inicialmente, el archivo expresiones.txt contiene lo siguiente:

```
b=7
a = 32.4 *(-8.6 - b)/       6.1E-8
d = a ^ b // Esto es un comentario
```

Y se espera como resultado:

```
Token                       Tipo
b                           VARIABLE
=                           ASIGNACION
7                           ENTERO
a                           VARIABLE
=                           ASIGNACION
32.4                        REAL
*                           MULTIPLICACION
(                           PARENTESIS_QUE_ABRE
-                           RESTA
8.6                         REAL
-                           RESTA
b                           VARIABLE
)                           PARENTESIS_QUE_CIERRA
/                           DIVISION
6.1E-8                      REAL
d                           VARIABLE
=                           ASIGNACION
a                           VARIABLE
^                           POTENCIA
b                           VARIABLE
// Esto es un comentario    COMENTARIO
```

Intenta probar con los siguientes casos, solo copia el texto de cada caso de prueba en el mismo archivo expresiones.txt y no te olvides de guardar el cambio

## Caso 1

```
//prueba 2
manzana =3
naranja_8=3.33E10 //falso
(direccion)/ 2. = a ^7var_2
```

*Resultado*

```
Token                   Tipo
//prueba 2              COMENTARIO
manzana                 VARIABLE
=                       ASIGNACION
3                       ENTERO
naranja_8               VARIABLE
=                       ASIGNACION
3.33E10                 REAL
//falso                 COMENTARIO
(                       PARENTESIS_QUE_ABRE
direccion               VARIABLE
)                       PARENTESIS_QUE_CIERRA
/                       DIVISION
2.                      REAL
=                       ASIGNACION
a                       VARIABLE
^                       POTENCIA
7var                    ERROR
_2                      ERROR
```

## Caso 2

```
tc = 22.7 / 3.3
te = (tc * 22) ^ 3
td = 33 // td es 33
```

*Resultado*

```
Token                   Tipo
tc                      VARIABLE
```

```
=                           ASIGNACION
22.7                        REAL
/                           DIVISION
3.3                         REAL
te                          VARIABLE
=                           ASIGNACION
(                           PARENTESIS_QUE_ABRE
tc                          VARIABLE
*                           MULTIPLICACION
22                          ENTERO
)                           PARENTESIS_QUE_CIERRA
^                           POTENCIA
3                           ENTERO
td                          VARIABLE
=                           ASIGNACION
33                          ENTERO
// td es 33                 COMENTARIO
```

**Caso 3**

```
.48 //estono escorrecto
0. * 56.33e_ flag -
56.33e3 / var_7
9.8e-10 +2 = grav
++1_ -7
```

*Resultado*

```
Token                       Tipo
.48                         ERROR
//estono escorrecto         COMENTARIO
0.                          REAL
*                           MULTIPLICACION
56.33e                      ERROR
_                           ERROR
flag                        VARIABLE
-                           RESTA
56.33e3                     REAL
/                           DIVISION
var_7                       VARIABLE
```

```
9.8e-10                     REAL
+                           SUMA
2                           ENTERO
=                           ASIGNACION
grav                        VARIABLE
+                           SUMA
+                           SUMA
1_                          ERROR
-                           RESTA
7                           ENTERO
```

**Caso 4**

```
var12_ / //primer comentario
12var //segundo comentario
var_12_7 * 33.1e10
```

*Resultado*

```
Token                       Tipo
var12_                      VARIABLE
/                           DIVISION
//primer comentario         COMENTARIO
12var                       ERROR
//segundo comentario        COMENTARIO
var_12_7                    VARIABLE
*                           MULTIPLICACION
33.1e10                     REAL
```

**Caso 5**

```
=int .8 // ups
/56/ 4.5
+* a_a7 ^3
prueba_esp          7 + 2//funciona
10. 10.0 _- 14.2E-10 1not ^
```

*Resultado*

```
Token                       Tipo
=                           ASIGNACION
int                         VARIABLE
.8                          ERROR
// ups                      COMENTARIO
/                           DIVISION
56                          ENTERO
/                           DIVISION
4.5                         REAL
+                           SUMA
*                           MULTIPLICACION
a_a7                        VARIABLE
^                           POTENCIA
3                           ENTERO
prueba_esp                  VARIABLE
7                           ENTERO
+                           SUMA
2                           ENTERO
//funciona                  COMENTARIO
10.                         REAL
10.0                        REAL
_                           ERROR
-                           RESTA
14.2E-10                    REAL
1not                        ERROR
^                           POTENCIA
```

**Caso 6**

```
1.1a1/ //esta mal
1.a*/
12.12a//hola
3.9E-8varbored+0./*
56.33e3/9.8e-10/2 = grav
```

*Resultado*

```
Token                       Tipo
1.1a1                       ERROR
/                           DIVISION
```

```
//esta mal                COMENTARIO
1.a                       ERROR
*                         MULTIPLICACION
/                         DIVISION
12.12a                    ERROR
//hola                    COMENTARIO
3.9E-8varbored            ERROR
+                         SUMA
0.                        REAL
/                         DIVISION
*                         MULTIPLICACION
56.33e3                   REAL
/                         DIVISION
9.8e-10                   REAL
/                         DIVISION
2                         ENTERO
=                         ASIGNACION
grav                      VARIABLE
```

## 3.4 Vídeo de demostración

Por último, aquí dejamos el link de un video de demostración .