Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Compiladores e intérpretes

Proyecto 1

Análisis Léxico

Profesor

Emmanuel Ramírez Segura

Subgrupo 5

Estudiantes

Antonio Fernández García - 2022075006

Marco Rodriguez Vargas - 2022149445

Josué Mena González - 2022138381

II Semestre 2024

Gramática

Lenguaje L1 – Especificación Formal

programa -> expresion ; { programa } ;
expresion -> identificador = expresion | termino { (+|-) termino } ;
termino -> factor { (*| /) factor } ;
factor -> identificador | numero | (expresion) ;
numero -> digito { digito } ;
digito -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Lenguaje L1 – Especificación Informal

- **Identificador**

Los identificadores son secuencias de una o más letras minúsculas, hasta un máximo de 12 letras.

- **{ }**

Los caracteres { y } se utilizan en las gramáticas formales para indicar repeticiones de elementos. En el contexto de las gramáticas de Backus-Naur (BNF) y otras notaciones similares, se utilizan para denotar que un elemento puede repetirse cero o más veces. Es equivalente a lo visto en la teoría de clase como (r)*. Por lo tanto no debe confundirse con símbolos terminales.

- **ws (White Spaces)**

Los espacios en blanco (tabs, espacios, enters) serán tomados como elementos separadores de los terminales. Será por lo tanto valido cualquiera de las siguientes notaciones:
✓ Válido: a=b+c;
✓ Válido: a = b + c ;
✓ Válido: a = b + c ;

Reconocimiento de Tokens

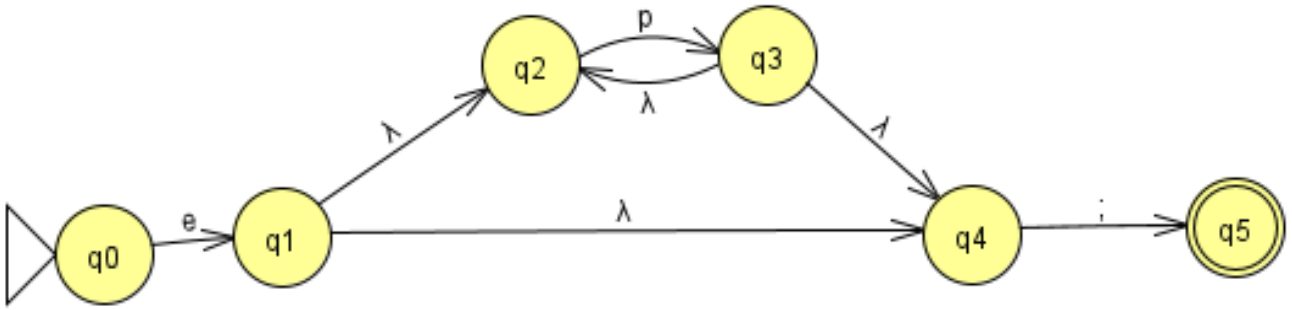
Expresión regular

1) programa -> expresion ; (programa)* ;
2) expresion -> (identificador = expresion | termino ((+ | -) termino)*) ;
3) termino -> factor((* | /) factor)* ;
4) factor -> identificador | numero | (expresion) ;
5) numero -> dígito+ ;
6) identificador -> letra+ ; (Máximo de 12)
7) digito -> [0-9]
8) letra -> [A-Za-z]
9) ws -> (blanco | tab | nuevalinea)+

AFN

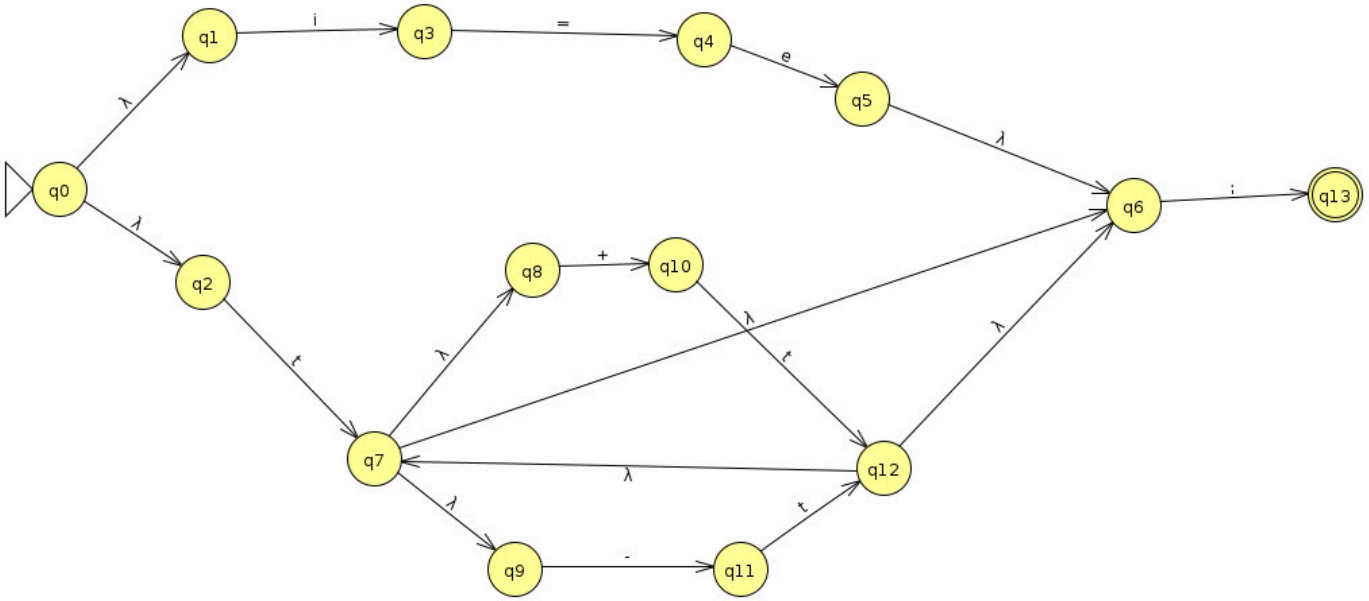
1) **Programa**

En este autómata, “p” representa un programa. Está compuesto por una expresión “e” seguida de un programa “p” y un punto y coma “;”, este último es un símbolo terminal. El programa puede repetirse cero o más veces gracias a la cerradura de Kleene



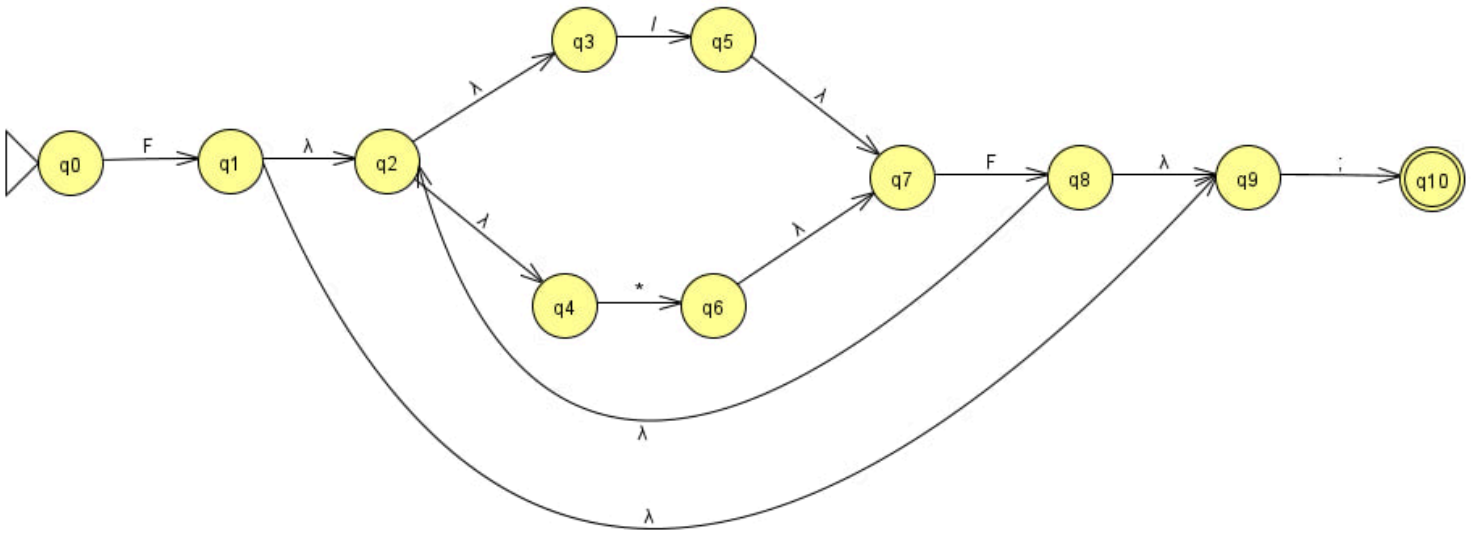
2) **Expresión**

En este autómata de expresión se toma la “i” como Identificador, la “e” como Expresión y “t” como Término (no se deben confundir con símbolos terminales).Los símbolos “=”, “+”, “-” y “;” son terminales.



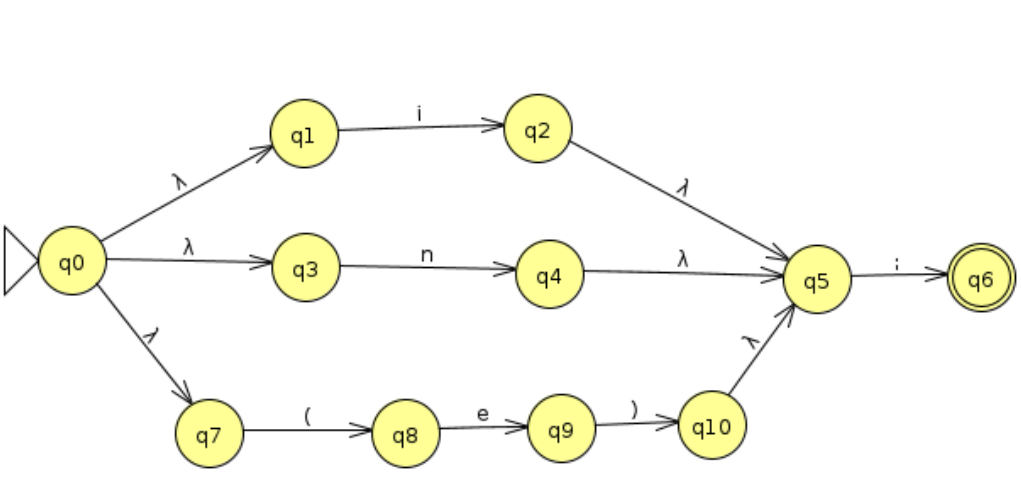
3) Término

En este autómata, “F” representa factor, “*” es el símbolo de producto, “/” símbolo de la división y “;” es el punto y coma.



4) Factor

En este autómata representa un factor, que puede ser un identificador “i”, un número “n” o una expresión “e” entre paréntesis. El punto y coma ”;” es terminal.

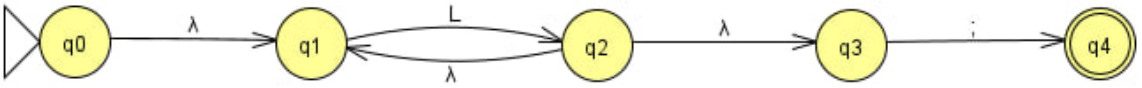


5) Número

En este caso es importante notar que la “d” actúa como dígito por las limitaciones del programa. No se debe confundir con otro símbolo que no sea un dígito del 0 al 9.

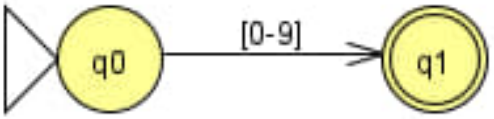
6) Identificador

En este autómata “L” representa letra, “;” es el punto y coma, además se debe aclarar que un



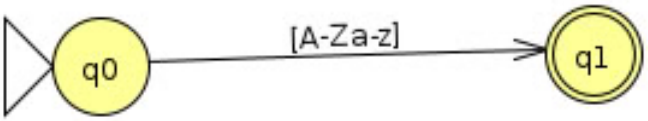
7) Dígito

El dígito puede ser cualquier número del 0 al 9, de ahí que la transición se representa como un [0-9]



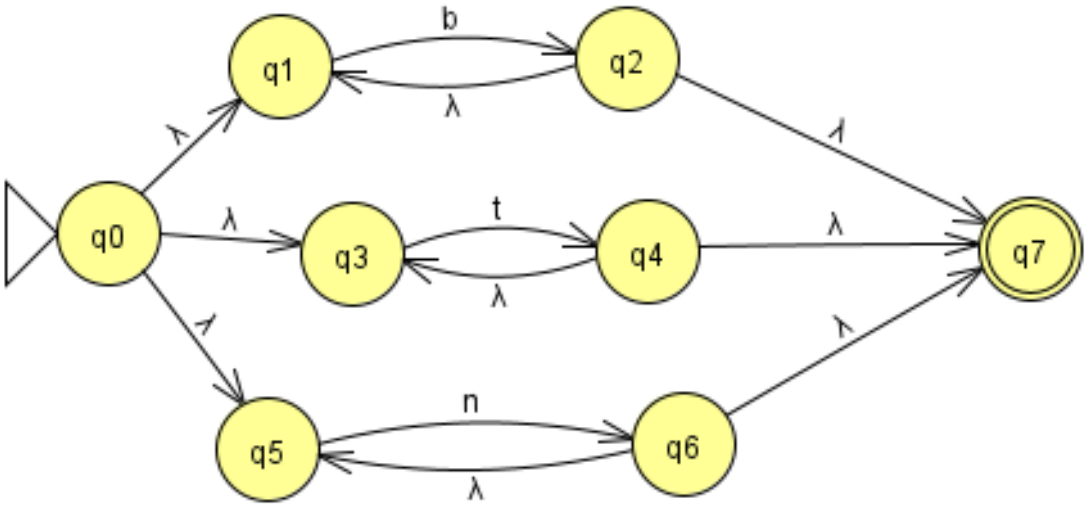
8) Letra

En este caso se toma cualquier letra de la A a la Z ya sea mayúscula o minúscula.



9) Ws

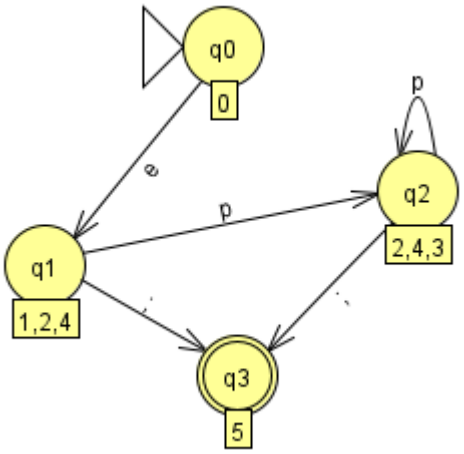
En este autómata, “b” representa el espacio en blanco (blank), “t” el símbolo de tabulación (tab) y “n” el símbolo de nueva línea (newline). Se aceptan secuencias que contienen uno o más de estos símbolos.



AFD

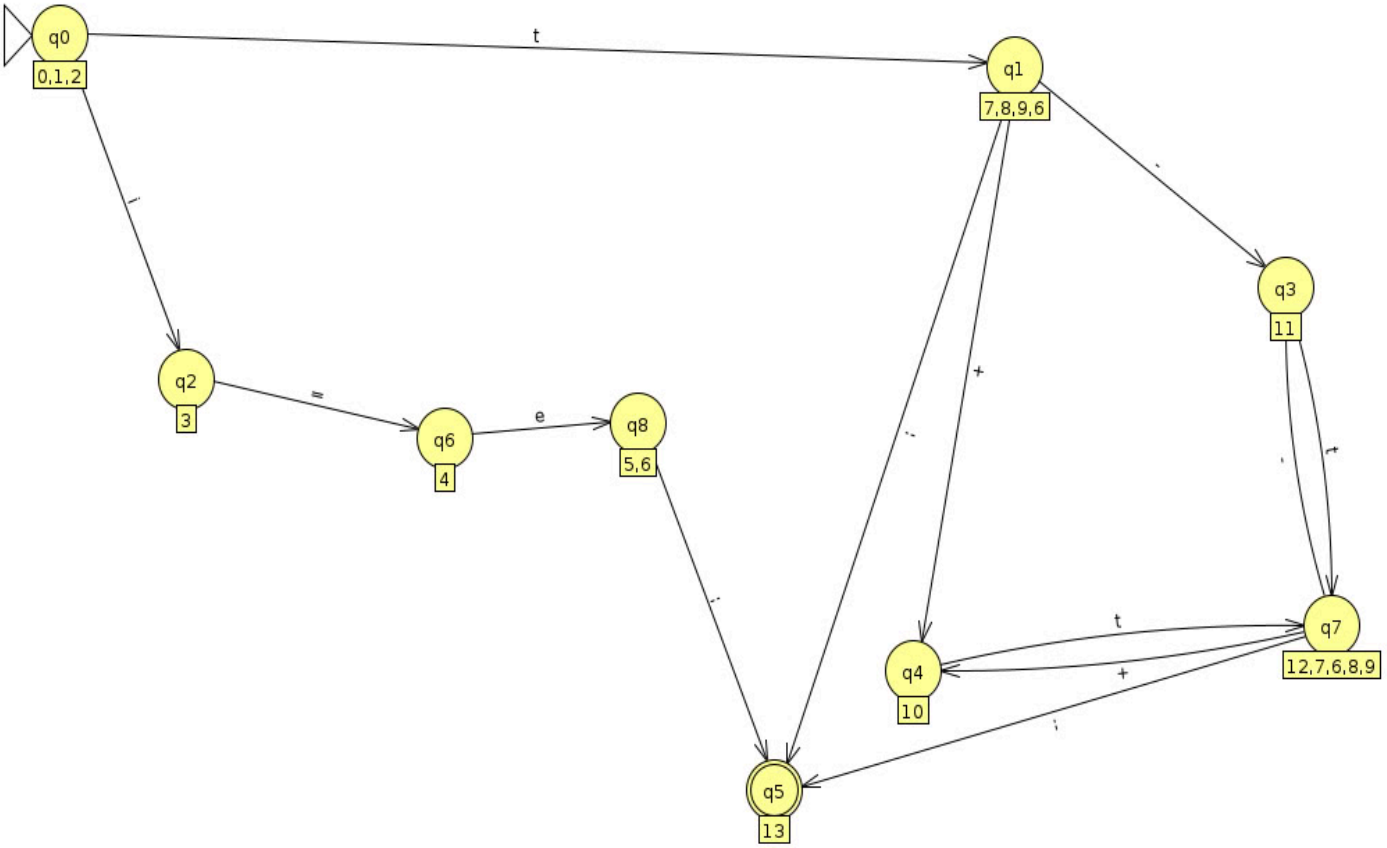
1) **Programa**

En este autómata, los puntos mencionados en el AFN también son válidos para este, ya que tiene la misma simbología.



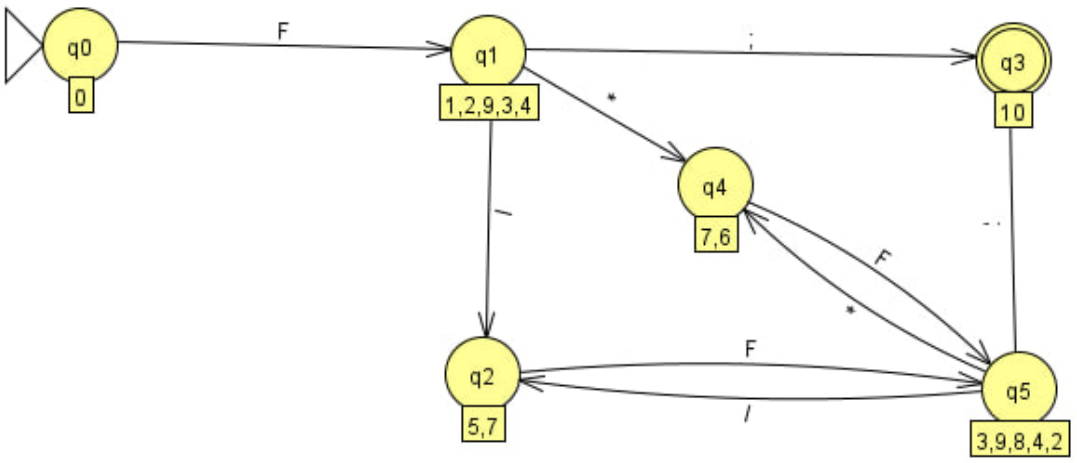
2) **Expresión**

Los puntos que se mencionaron en el AFN también son válidos para este, ya que tiene la misma simbología.



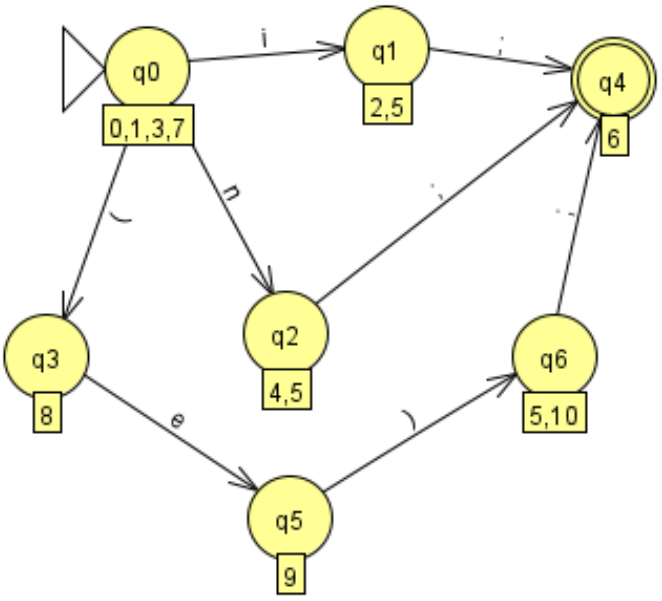
3) Término

Al igual que en el AFN, “F” representa factor, “*” es el símbolo de producto, “/” símbolo de la división y “;” es el punto y coma.



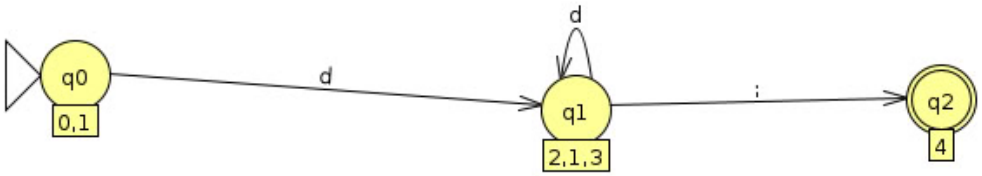
4) Factor

Los puntos mencionados en el AFN también son válidos para este AFD, ya que tiene la misma simbología.



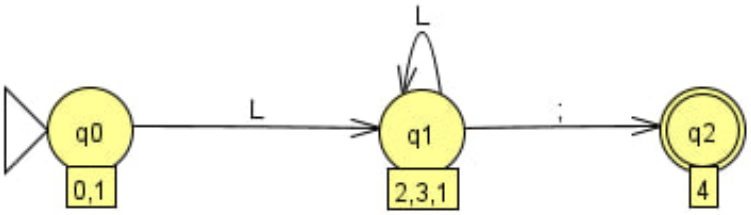
5) **Número**

En este caso la d sigue representando un dígito del 0 al 9.



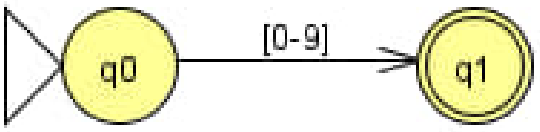
6) **Identificador**

Al igual que cuando era un AFN, “L” representa letra, “;” punto y coma y el identificador no puede tener más de 12 letras



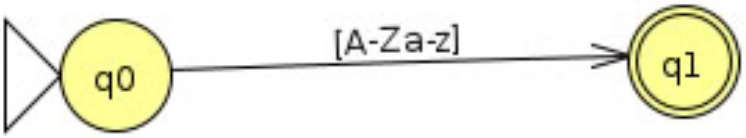
7) **Dígito**

Al ser un autómata tan sencillo su versión AFN y AFD es la misma



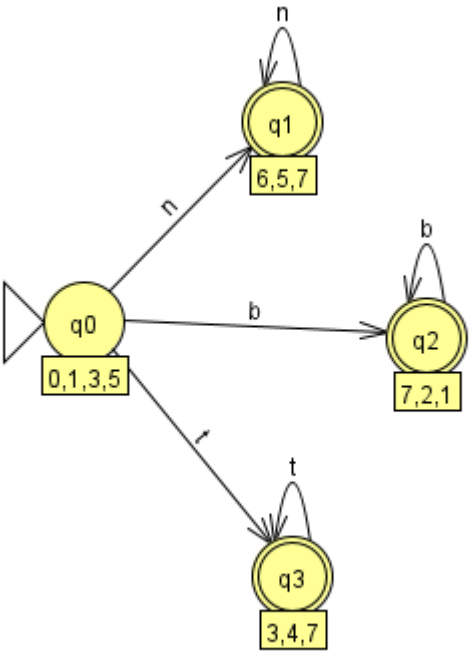
8) **Letra**

Este ya es un AFD simplificado al máximo



9) Ws

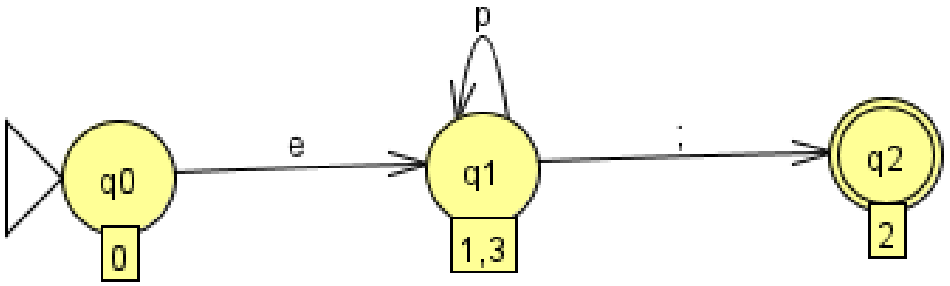
Los puntos mencionados en el AFN también son válidos para este AFD, ya que tiene la misma simbología y reconoce las mismas secuencias de espacios en blanco, tabulaciones y nuevas líneas.



Simplificación del AFN

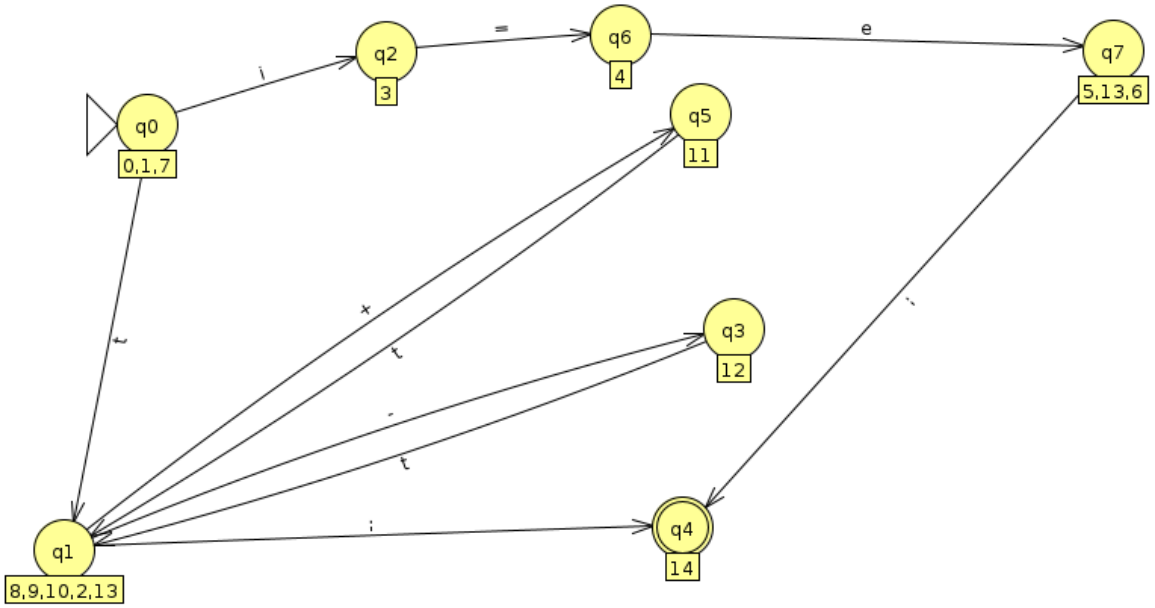
1) **Programa**

En este autómata, se unificaron los estados de “e” antes de “p”, ya que conducían al mismo estado. Al final, todas las transiciones llevan al símbolo terminal “;” que conduce al estado final



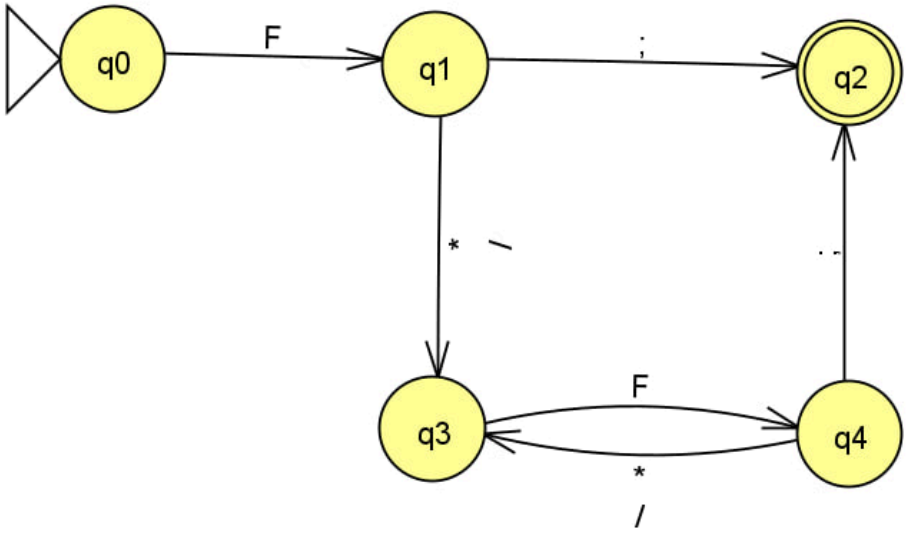
2) **Expresión**

Para este punto se lograron reducir ciertas transiciones y se acortó un estado total.



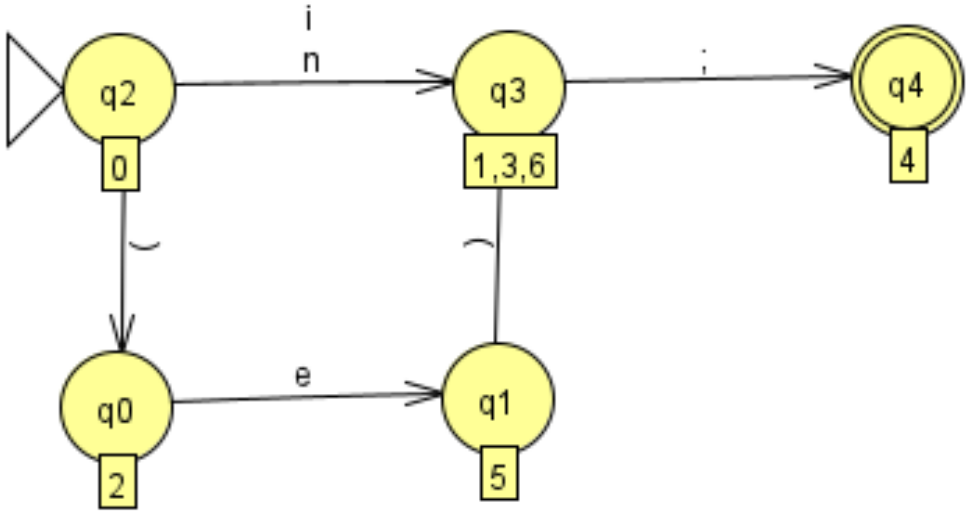
3) Término

Para reducir un estado, se unifica las transiciones de “*” y “/” pues en ambos casos se llega al mismo lugar, reduciendo su tamaño pero manteniendo su comportamiento. “F” es factor, “*” producto, “/” división y “;” punto y coma



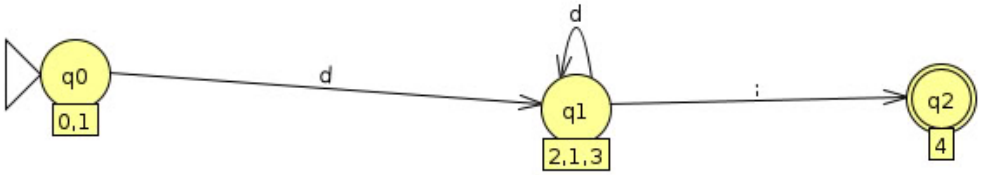
4) Factor

En este autómata se lograron unificar las transiciones de “i” (identificador) y “n” (número), ya que ambas conducían al mismo estado. Esto permitió reducir el número de estados y simplificar el AFD. El ”;” es terminal y conduce al estado de aceptación.



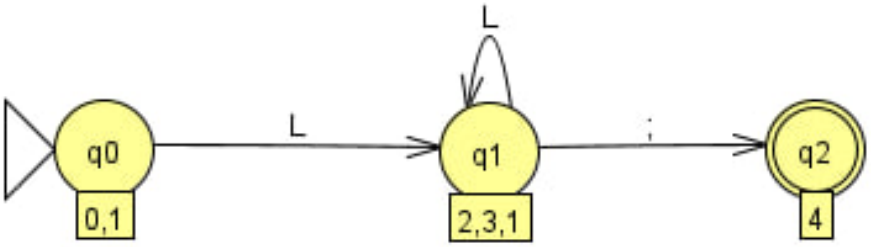
5) **Número**

En este caso por error de la aplicación y porque este ya es bastante pequeño sigue siendo el mismo AFD. No se reduce.



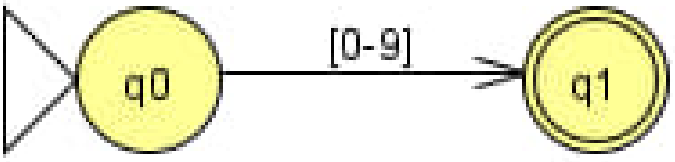
6) **Identificador**

Ya se encuentra simplificada al máximo



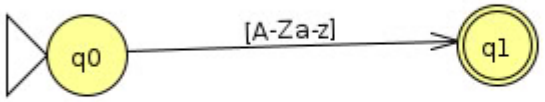
7) **Dígito**

Ya se encuentra simplificada al máximo



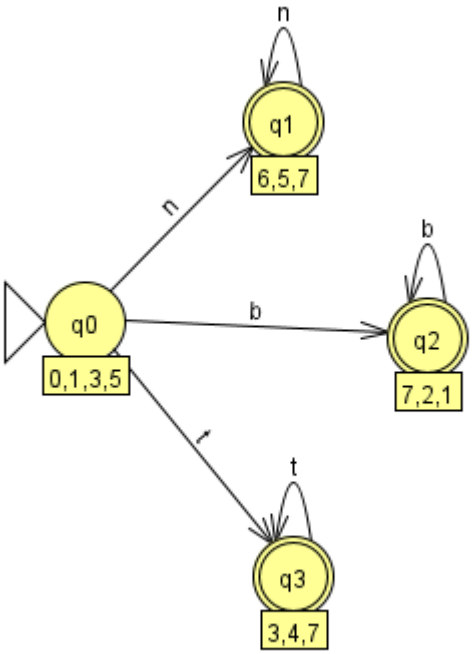
8) **Letra**

Este se mantiene igual ya que está simplificado al máximo.



9) **Ws**

Este se mantiene igual ya que está simplificado al máximo.



Pruebas funcionales

Ejemplos sin errores léxicos y evidencia de la tokenización con el lexer

Ejemplo 1

a = (x + 2) * 3;
b = c / (5 + d);
e = f + g * h;

En la primera línea, a = (x + 2) * 3;, el identificador a está a la izquierda de una asignación. La expresión a la derecha es válida porque x es un identificador, 2 es un número y la operación de suma está correctamente agrupada entre paréntesis. Por último la multiplicación por 3 sigue las reglas de la gramática.

En la segunda línea, b = c / (5 + d);, b es un identificador. La expresión a la derecha es válida ya que c y d son identificadores, 5 es un número y la suma está correctamente agrupada dentro de paréntesis seguida por la división.

En la tercera línea, e = f + g * h;, e es un identificador. La expresión a la derecha es correcta, con una suma entre f y el producto de g y h, cumpliendo las reglas de precedencia y operaciones válidas.

Todas las líneas terminan correctamente con punto y coma.

```
C:\Users\User\CompiProyecto1\miCompilador>mvn exec:java -Dexec
.args="lexCorrecto1.txt"
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< com.miCompilador:miCompilador >---
----------------
[INFO] Building miCompilador 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------
----------------
[INFO]
[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ miComp
ilador ---
```

```
Valor: a, Tipo: IDENTIFICADOR
Valor: =, Tipo: ASIGNACION
Valor: (, Tipo: PARENTESIS_IZQ
Valor: x, Tipo: IDENTIFICADOR
Valor: +, Tipo: SUMA
Valor: 2, Tipo: NUMERO
Valor: ), Tipo: PARENTESIS_DER
Valor: *, Tipo: MULTIPLICACION
Valor: 3, Tipo: NUMERO
Valor: ;, Tipo: PUNTO_COMA
Valor: b, Tipo: IDENTIFICADOR
Valor: =, Tipo: ASIGNACION
Valor: c, Tipo: IDENTIFICADOR
Valor: /, Tipo: DIVISION
Valor: (, Tipo: PARENTESIS_IZQ
Valor: 5, Tipo: NUMERO
Valor: +, Tipo: SUMA
Valor: d, Tipo: IDENTIFICADOR
Valor: ), Tipo: PARENTESIS_DER
Valor: ;, Tipo: PUNTO_COMA
Valor: e, Tipo: IDENTIFICADOR
Valor: =, Tipo: ASIGNACION
Valor: f, Tipo: IDENTIFICADOR
Valor: +, Tipo: SUMA
Valor: g, Tipo: IDENTIFICADOR
Valor: *, Tipo: MULTIPLICACION
Valor: h, Tipo: IDENTIFICADOR
Valor: ;, Tipo: PUNTO_COMA
Nombre token: a
Informacion:
Tipo: entero
Ambito: global
N├╢mero de linea: 1

Nombre token: b
Informacion:
Tipo: entero
Ambito: global
N├╢mero de linea: 2

Nombre token: e
Informacion:
Tipo: entero
Ambito: global
N├╢mero de linea: 3

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.345 s
[INFO] Finished at: 2024-09-23T02:19:57-06:00
[INFO] ------------------------------------------------------------------------
'cmd' is not recognized as an internal or external command,
operable program or batch file.
```

Ejemplo 2

t = (a + (b - c)) * d;
u = (x / (y * 2)) + (z - 4);
m = n * 5 + 7 / k;

En la primera línea, t = (a + (b - c)) * d;, t es un identificador. La expresión a la derecha es válida porque a, b, c, y d son identificadores, la suma y resta están correctamente agrupadas entre paréntesis, y luego se realiza la multiplicación por d.

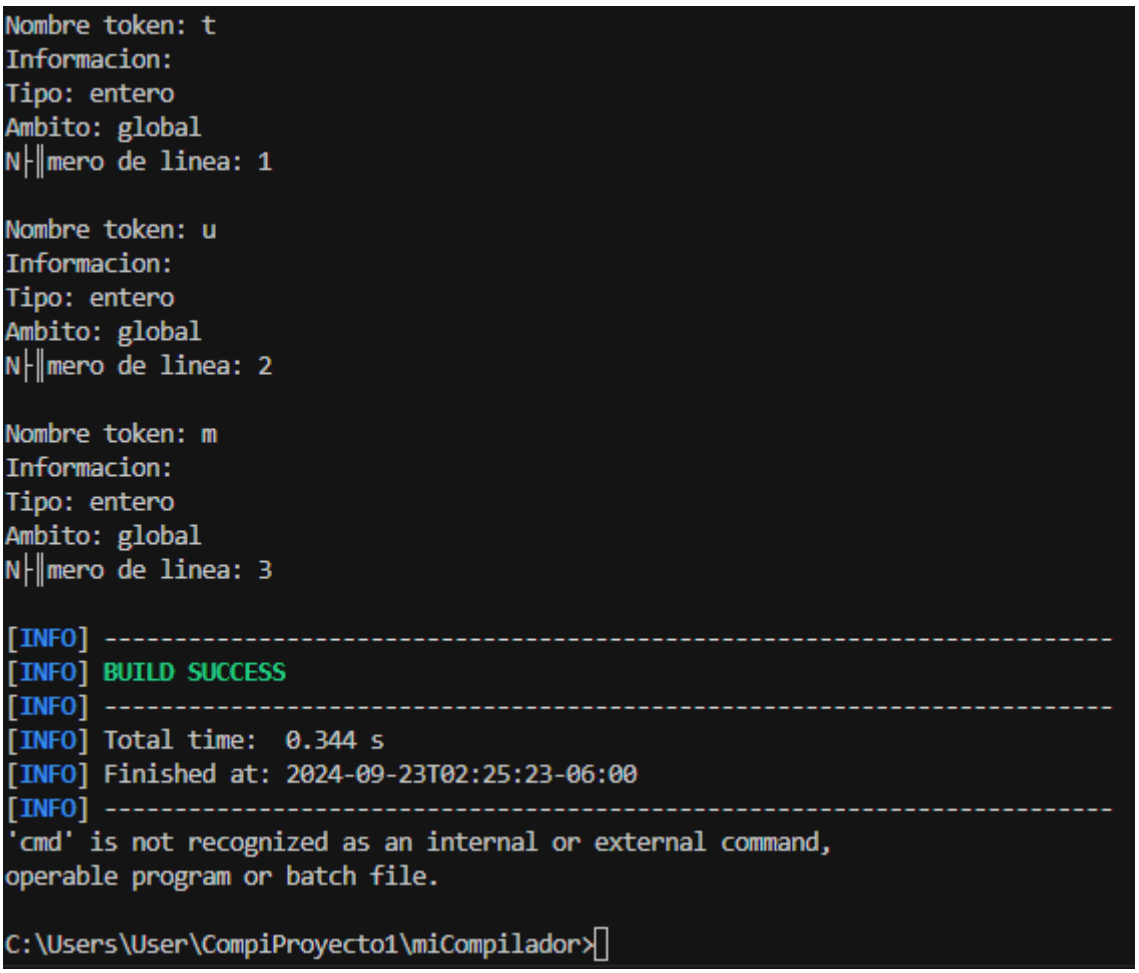
En la segunda línea, u = (x / (y * 2)) + (z - 4);, u es un identificador. La expresión a la derecha es válida, con la división de x por el producto de y y 2 agrupada entre paréntesis, seguida de la suma con la resta de z y 4.

En la tercera línea, m = n * 5 + 7 / k;, m es un identificador. La expresión es correcta, con la multiplicación de n por 5 y la suma con el resultado de 7 entre k.

Todas las líneas terminan correctamente con punto y coma.

```
C:\Users\User\CompiProyecto1\miCompilador>mvn exec:java -Dexec.args="lexCorrecto2.txt"
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< com.miCompilador:miCompilador >--------------------
[INFO] Building miCompilador 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ miCompilador ---
```

```
Valor: t, Tipo: IDENTIFICADOR
Valor: =, Tipo: ASIGNACION
Valor: (, Tipo: PARENTESIS_IZQ
Valor: a, Tipo: IDENTIFICADOR
Valor: +, Tipo: SUMA
Valor: (, Tipo: PARENTESIS_IZQ
Valor: b, Tipo: IDENTIFICADOR
Valor: -, Tipo: RESTA
Valor: c, Tipo: IDENTIFICADOR
Valor: ), Tipo: PARENTESIS_DER
Valor: ), Tipo: PARENTESIS_DER
Valor: *, Tipo: MULTIPLICACION
Valor: d, Tipo: IDENTIFICADOR
Valor: ;, Tipo: PUNTO_COMA
Valor: u, Tipo: IDENTIFICADOR
Valor: =, Tipo: ASIGNACION
Valor: (, Tipo: PARENTESIS_IZQ
Valor: x, Tipo: IDENTIFICADOR
Valor: /, Tipo: DIVISION
Valor: (, Tipo: PARENTESIS_IZQ
Valor: y, Tipo: IDENTIFICADOR
Valor: *, Tipo: MULTIPLICACION
Valor: 2, Tipo: NUMERO
Valor: ), Tipo: PARENTESIS_DER
Valor: ), Tipo: PARENTESIS_DER
Valor: +, Tipo: SUMA
Valor: (, Tipo: PARENTESIS_IZQ
Valor: z, Tipo: IDENTIFICADOR
Valor: -, Tipo: RESTA
Valor: 4, Tipo: NUMERO
Valor: ), Tipo: PARENTESIS_DER
Valor: ;, Tipo: PUNTO_COMA
Valor: m, Tipo: IDENTIFICADOR
Valor: =, Tipo: ASIGNACION
Valor: n, Tipo: IDENTIFICADOR
Valor: *, Tipo: MULTIPLICACION
Valor: 5, Tipo: NUMERO
Valor: +, Tipo: SUMA
Valor: 7, Tipo: NUMERO
Valor: /, Tipo: DIVISION
Valor: k, Tipo: IDENTIFICADOR
Valor: ;, Tipo: PUNTO_COMA
```


Nombre token: t
Informacion:
Tipo: entero
Ambito: global
N├╢mero de linea: 1
Nombre token: u
Informacion:
Tipo: entero
Ambito: global
N├╢mero de linea: 2
Nombre token: m
Informacion:
Tipo: entero
Ambito: global
N├╢mero de linea: 3
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 0.344 s
[INFO] Finished at: 2024-09-23T02:25:23-06:00
[INFO] --
'cmd' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\User\CompiProyecto1\miCompilador>

Ejemplos con errores léxicos y evidencia de cómo se maneja el error

Ejemplo 1

a = 5 + 3;
b = 9 / (2 *$ 4);
x y = 10:

En la primera línea, a = 5 + 3;, es correcta ya que a es un identificador válido, los números y el operador de suma son correctos, y la línea termina con punto y coma.

En la segunda línea, b = 9 / (2 *$ 4);, es incorrecta porque el símbolo $ no es un operador permitido según la gramática.

En la tercera línea, x y = 10:, es incorrecta porque la línea no termina con punto y coma sino con un signo de dos puntos lo cual no está permitido.

```
C:\Users\User\CompiProyecto1\miCompilador>mvn exec:java -Dexec.args="lexError1.txt"
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< com.miCompilador:miCompilador >--------------------
[INFO] Building miCompilador 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ miCompilador ---
Valor: a, Tipo: IDENTIFICADOR
Valor: =, Tipo: ASIGNACION
Valor: 5, Tipo: NUMERO
Valor: +, Tipo: SUMA
Valor: c, Tipo: IDENTIFICADOR
Valor: 3, Tipo: NUMERO
Valor: ;, Tipo: PUNTO_COMA
Valor: b, Tipo: IDENTIFICADOR
Valor: =, Tipo: ASIGNACION
Valor: 9, Tipo: NUMERO
Valor: /, Tipo: DIVISION
Valor: (, Tipo: PARENTESIS_IZQ
Valor: 2, Tipo: NUMERO
Valor: *, Tipo: MULTIPLICACION
 Error [Fase Le╠üxica]: La l├¡nea 2 error, lexema no reconocido: $
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.348 s
[INFO] Finished at: 2024-09-23T02:35:51-06:00
[INFO] ------------------------------------------------------------------------
'cmd' is not recognized as an internal or external command,
operable program or batch file.
```

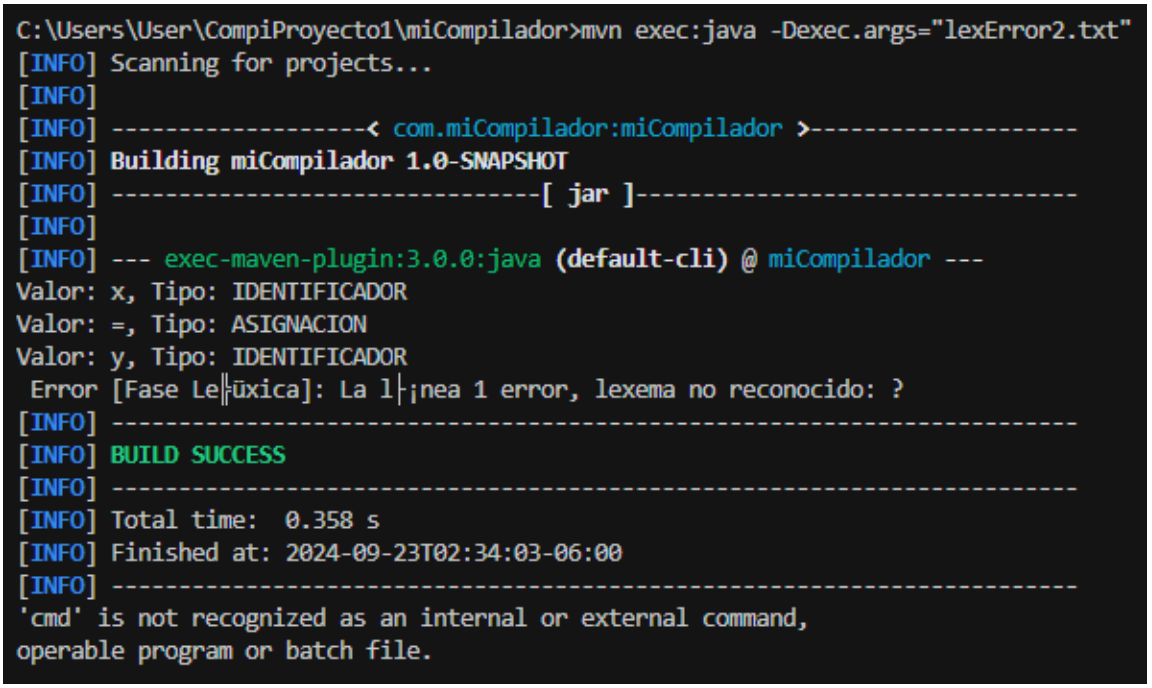
Ejemplo 2

x = y ? 2;
f = 7 + #;
n @ b = 12 - 5;

En la primera línea, x = y ? 2;, es incorrecta porque el símbolo ? no es un operador válido según la gramática.

En la segunda línea, f = 7 + #;, es incorrecta ya que el símbolo # no es un identificador ni un número permitido en la gramática.

En la tercera línea, n @ b = 12 - 5;, es incorrecta porque el símbolo @ no es un operador válido



Referencias

Aho, A., Lam, M., Sethi, R., & Ullman, J. (2006). Compilers: Principles, techniques and tools (2nd. Edition ed.). Addison-Wesley.