



Instituto Tecnológico de Costa Rica

Escuela de computación

Compiladores E Intérpretes GR20

Avance 1 Proyecto#1 Compiladores e intérpretes

Estudiante:

Brandon Rodríguez Vega - 2021152632

Mariano Oconitrillo Vega- 2021080289

Sigifredo Chacón Balladares – 2021067590

Sub-grupo: 2

Profesor:

Roger Emmanuel Ramirez Segura

Centro Académico de Alajuela

Semestre I

2023

Índice

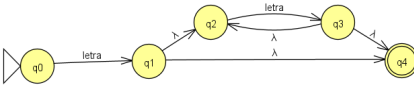
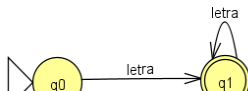
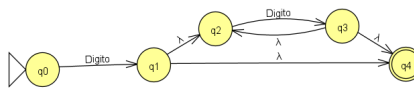
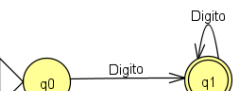
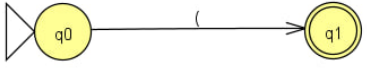
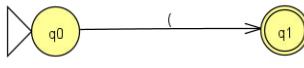
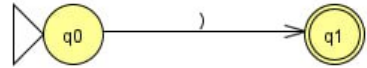
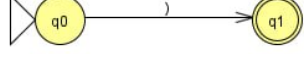

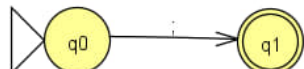
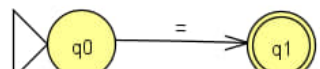
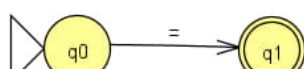
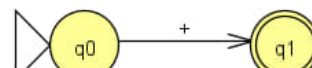
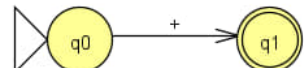
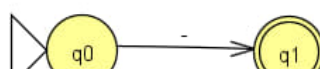



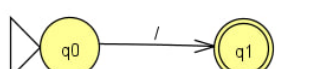
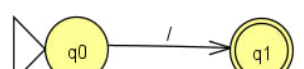
Gramática:	2
RECONOCIMIENTO DE TOKENS:	3
Implementaciones de software de cada autómata:	4
• IDENTIFICADOR:	4
• NUMERO	4
• PARENTESIS_IZQ	5
• PARENTESIS_DER	5
• PUNTO_COMA	6
• ASIGNACION	6
• SUMA	7
• RESTA	7
• MULTIPLICACION	8
• DIVISION	8
Pruebas Funcionales:	9
Ejemplo 1:	9
Ejemplo 2:	10
Ejemplo 3:	11
Ejemplo 4:	13

Gramática:

Lenguaje L1

```
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
```

RECONOCIMIENTO DE TOKENS:

Valor	Tipo	Expresión Regular	AFN	AFD
variable	IDENTIFICADOR	[a-z]	 <p>Nota: Solo se pueden repetir un máximo de 12 veces.</p>	
1234	NUMERO	[0-9]		
(PARENTESIS_IZQ	[(]		
)	PARENTESIS_DER	[)]		
;	PUNTO_COMA	[;]		
=	ASIGNACION	[=]		
+	SUMA	[+]		
-	RESTA	[-]		
*	MULTIPLICACIÓN	[*]		
/	DIVISION	[/]		

Implementaciones de software de cada autómata:

- IDENTIFICADOR:

```
else if (Character.isLetter(caracter) && Character.isLowerCase(caracter) ) {  
    lexema.append(caracter);  
    estado = Estados.IDENTIFICADOR;  
}
```

```
case IDENTIFICADOR:  
    if(Character.isWhitespace(caracter)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"IDENTIFICADOR"));  
        hashMap.put(lexema.toString(), "IDENTIFICADOR");  
        lexema.setLength(0);  
        continue;  
    }  
    else if(Character.isLetter(caracter) && Character.isLowerCase(caracter) && lexema.length() < 12){  
        lexema.append(caracter);  
    }  
  
    else {  
        estado = Estados.Error;  
        lexema.append(caracter);  
    }  
    break;
```

- NUMERO

```
else if (Character.isDigit(caracter)) {  
    lexema.append(caracter);  
    estado = Estados.NUMERO;  
}
```

```
case NUMERO:  
    if(Character.isWhitespace(caracter)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"NUMERO"));  
        lexema.setLength(0);  
        continue;  
    }  
    else if(Character.isDigit(caracter)){  
        lexema.append(caracter);  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(caracter);  
    }  
    break;
```

- PARENTESIS_IZQ

```
else if (caracter == '(') {  
    lexema.append(caracter);  
    estado = Estados.PARENTESIS_IZQ;  
}
```

```
case PARENTESIS_IZQ:  
    if(Character.isWhitespace(caracter)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"PARENTESIS_IZQ"));  
        lexema.setLength(0);  
        continue;  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(caracter);  
    }  
    break;
```

- PARENTESIS_DER

```
else if (caracter == ')') {  
    lexema.append(caracter);  
    estado = Estados.PARENTESIS_DER;  
}
```

```
case PARENTESIS_DER:  
    if(Character.isWhitespace(caracter)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"PARENTESIS_DER"));  
        lexema.setLength(0);  
        continue;  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(caracter);  
    }  
    break;
```

- PUNTO_COMA

```
else if (character == ';') {  
    lexema.append(character);  
    estado = Estados.PUNTO_COMA;  
}
```

```
case PUNTO_COMA:  
    if(Character.isWhitespace(character)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"PUNTO_COMA"));  
        lexema.setLength(0);  
        continue;  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(character);  
    }  
    break;
```

- ASIGNACION

```
else if (character == '=') {  
    lexema.append(character);  
    estado = Estados.ASIGNACION;  
}
```

```
case ASIGNACION:  
    if(Character.isWhitespace(character)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"ASIGNACION"));  
        lexema.setLength(0);  
        continue;  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(character);  
    }  
    break;
```

- SUMA

```
else if (caracter == '+') {  
    lexema.append(caracter);  
  
    estado = Estados.SUMA;  
}
```

```
case SUMA:  
    if(Character.isWhitespace(caracter)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"SUMA"));  
        lexema.setLength(0);  
        continue;  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(caracter);  
    }  
    break;
```

- RESTA

```
else if (caracter == '-') {  
    lexema.append(caracter);  
    estado = Estados.RESTA;  
}
```

```
case RESTA:  
    if(Character.isWhitespace(caracter)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"RESTA"));  
        lexema.setLength(0);  
        continue;  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(caracter);  
    }  
    break;
```


- MULTIPLICACION

```
else if (caracter == '*') {  
    lexema.append(caracter);  
    estado = Estados.MULTIPLICACION;  
}
```

```
case MULTIPLICACION:  
    if(Character.isWhitespace(caracter)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"MULTIPLICACION"));  
        lexema.setLength(0);  
        continue;  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(caracter);  
    }  
    break;
```

- DIVISION

```
else if (caracter == '/') {  
    lexema.append(caracter);  
    estado = Estados.DIVISION;  
}
```

```
case DIVISION:  
    if(Character.isWhitespace(caracter)){  
        estado = Estados.INICIO;  
        listaToken.add(new Token(lexema.toString(),"DIVISION"));  
        lexema.setLength(0);  
        continue;  
    }  
    else {  
        estado = Estados.Error;  
        lexema.append(caracter);  
    }  
    break;
```

Pruebas Funcionales:

Ejemplo 1:

Vamos a probar la siguiente entrada:

```
a = 42 ;  
b = a - 7 ;  
c = ( a + b ) * 2 ;
```

En este ejemplo podemos ver 3 líneas de código donde hay 3 identificadores los cuales son: “a”, “b”, “c”. Luego tenemos tres signos de igual “=”, también tenemos los números 42, el 7 y el 2. También tenemos los símbolos de resta “-”, el símbolo de suma “+”, el símbolo de multiplicación “*”, el símbolo de punto y coma “;” y los símbolos de paréntesis izquierdo y derecho “()”.

Cuando ejecutamos el programa, nos da el siguiente archivo de salida:

```
1 Token: a, Tipo: IDENTIFICADOR  
2 Token: =, Tipo: ASIGNACION  
3 Token: 42, Tipo: NUMERO  
4 Token: ;, Tipo: PUNTO_COMA  
5 Token: b, Tipo: IDENTIFICADOR  
6 Token: =, Tipo: ASIGNACION  
7 Token: a, Tipo: IDENTIFICADOR  
8 Token: -, Tipo: RESTA  
9 Token: 7, Tipo: NUMERO  
10 Token: ;, Tipo: PUNTO_COMA  
11 Token: c, Tipo: IDENTIFICADOR  
12 Token: =, Tipo: ASIGNACION  
13 Token: (, Tipo: PARENTESIS_IZQ  
14 Token: a, Tipo: IDENTIFICADOR  
15 Token: +, Tipo: SUMA  
16 Token: b, Tipo: IDENTIFICADOR  
17 Token: ), Tipo: PARENTESIS_DER  
18 Token: *, Tipo: MULTIPLICACION  
19 Token: 2, Tipo: NUMERO  
20 Token: ;, Tipo: PUNTO_COMA  
21
```

El archivo nos muestra todos los tokens que reconoció el lexer, se encontraron 20 tokens, cada token nos muestra el valor del token y su tipo, el archivo de entrada no contiene errores por lo que se reconoció un token para cada uno de los elementos antes mencionados de la entrada

Ejemplo 2:

Vamos a probar la siguiente entrada:

```
args = 13 ;  
os = ( bbt * xyz ) + 76 ;  
ppth = args - 20 / radio ;
```

En este ejemplo podemos ver 3 líneas de código donde hay 6 identificadores: “args”, “os”, “bbt”, “xyz”, “ppth” y “radio”. También se encuentran varios números como: el 13, el 76 y el 20. Se tienen también distintos símbolos como el de asignación “=”, los paréntesis izquierdo y derecho “()”, el signo de multiplicación “*”, el signo de suma “+”, el símbolo de punto y coma “;”, el signo de resta “-” y el símbolo de división “/”.

Cuando ejecutamos el programa, nos da el siguiente archivo de salida:

```
1 Token: args, Tipo: IDENTIFICADOR  
2 Token: =, Tipo: ASIGNACION  
3 Token: 13, Tipo: NUMERO  
4 Token: ;, Tipo: PUNTO_COMA  
5 Token: os, Tipo: IDENTIFICADOR  
6 Token: =, Tipo: ASIGNACION  
7 Token: (, Tipo: PARENTESIS_IZQ  
8 Token: bbt, Tipo: IDENTIFICADOR  
9 Token: *, Tipo: MULTIPLICACION  
10 Token: xyz, Tipo: IDENTIFICADOR  
11 Token: ), Tipo: PARENTESIS_DER  
12 Token: +, Tipo: SUMA  
13 Token: 76, Tipo: NUMERO  
14 Token: ;, Tipo: PUNTO_COMA  
15 Token: ppth, Tipo: IDENTIFICADOR  
16 Token: =, Tipo: ASIGNACION  
17 Token: args, Tipo: IDENTIFICADOR  
18 Token: -, Tipo: RESTA  
19 Token: 20, Tipo: NUMERO  
20 Token: /, Tipo: DIVISION  
21 Token: radio, Tipo: IDENTIFICADOR  
22 Token: ;, Tipo: PUNTO_COMA  
23
```

El archivo nos muestra todos los tokens que reconoció el lexer, se encontraron 22 tokens, cada token nos muestra el valor del token y su tipo, el archivo de entrada no contiene errores por lo que se reconoció un token para cada uno de los elementos antes mencionados de la entrada

Ejemplo 3:

Vamos a probar la siguiente entrada:

```
x = 10 ;
```

```
y = 20 ;
```

```
z = 30 ;
```

```
a = x + y ;
```

```
b = y - z ;
```

```
c = z * x ;
```

```
d = x / y ;
```

```
f = ( a + b ) * c ;
```

```
g = ( d - e ) * ( x + y ) ;
```

```
i = ( h + g ) * ( d - c ) ;
```

```
j = ( i - h ) / ( g + f ) ;
```

En este ejemplo tenemos 13 líneas de código, contando las líneas en blanco, en las cuales tenemos 13 identificadores los cuales van de la “ a “ hasta la “ j “ y luego la “ x “, “ y “, “ z “. También se tienen varios símbolos como el de asignación “=”, el punto y coma “;”, la suma “+”, la resta “-”, la multiplicación “*“, la división “/” y los paréntesis izquierdo y derecho “ () “.

Cuando ejecutamos el programa, nos da el siguiente archivo de salida:

```
1 Token: x, Tipo: IDENTIFICADOR
2 Token: =, Tipo: ASIGNACION
3 Token: 10, Tipo: NUMERO
4 Token: ,, Tipo: PUNTO_COMA
5 Token: y, Tipo: IDENTIFICADOR
6 Token: =, Tipo: ASIGNACION
7 Token: 20, Tipo: NUMERO
8 Token: ,, Tipo: PUNTO_COMA
9 Token: z, Tipo: IDENTIFICADOR
10 Token: =, Tipo: ASIGNACION
11 Token: 30, Tipo: NUMERO
12 Token: ,, Tipo: PUNTO_COMA
13 Token: a, Tipo: IDENTIFICADOR
14 Token: =, Tipo: ASIGNACION
15 Token: x, Tipo: IDENTIFICADOR
16 Token: +, Tipo: SUMA
17 Token: y, Tipo: IDENTIFICADOR
18 Token: ,, Tipo: PUNTO_COMA
19 Token: b, Tipo: IDENTIFICADOR
20 Token: =, Tipo: ASIGNACION
21 Token: y, Tipo: IDENTIFICADOR
22 Token: -, Tipo: RESTA
23 Token: z, Tipo: IDENTIFICADOR
24 Token: ,, Tipo: PUNTO_COMA
25 Token: c, Tipo: IDENTIFICADOR
26 Token: =, Tipo: ASIGNACION
27 Token: z, Tipo: IDENTIFICADOR
28 Token: *, Tipo: MULTIPLICACION
29 Token: x, Tipo: IDENTIFICADOR
30 Token: ,, Tipo: PUNTO_COMA
31 Token: d, Tipo: IDENTIFICADOR
32 Token: =, Tipo: ASIGNACION
33 Token: x, Tipo: IDENTIFICADOR
34 Token: /, Tipo: DIVISION
35 Token: y, Tipo: IDENTIFICADOR
36 Token: ,, Tipo: PUNTO_COMA
37 Token: f, Tipo: IDENTIFICADOR
38 Token: =, Tipo: ASIGNACION
39 Token: (, Tipo: PARENTESIS_IZQ
40 Token: a, Tipo: IDENTIFICADOR
```

```
48 Token: =, Tipo: ASIGNACION
49 Token: (, Tipo: PARENTESIS_IZQ
50 Token: d, Tipo: IDENTIFICADOR
51 Token: -, Tipo: RESTA
52 Token: e, Tipo: IDENTIFICADOR
53 Token: ), Tipo: PARENTESIS_DER
54 Token: *, Tipo: MULTIPLICACION
55 Token: (, Tipo: PARENTESIS_IZQ
56 Token: x, Tipo: IDENTIFICADOR
57 Token: +, Tipo: SUMA
58 Token: y, Tipo: IDENTIFICADOR
59 Token: ), Tipo: PARENTESIS_DER
60 Token: ,, Tipo: PUNTO_COMA
61 Token: i, Tipo: IDENTIFICADOR
62 Token: =, Tipo: ASIGNACION
63 Token: (, Tipo: PARENTESIS_IZQ
64 Token: h, Tipo: IDENTIFICADOR
65 Token: +, Tipo: SUMA
66 Token: g, Tipo: IDENTIFICADOR
67 Token: ), Tipo: PARENTESIS_DER
68 Token: *, Tipo: MULTIPLICACION
69 Token: (, Tipo: PARENTESIS_IZQ
70 Token: d, Tipo: IDENTIFICADOR
71 Token: -, Tipo: RESTA
72 Token: c, Tipo: IDENTIFICADOR
73 Token: ), Tipo: PARENTESIS_DER
74 Token: ,, Tipo: PUNTO_COMA
75 Token: j, Tipo: IDENTIFICADOR
76 Token: =, Tipo: ASIGNACION
77 Token: (, Tipo: PARENTESIS_IZQ
78 Token: i, Tipo: IDENTIFICADOR
79 Token: -, Tipo: RESTA
80 Token: h, Tipo: IDENTIFICADOR
81 Token: ), Tipo: PARENTESIS_DER
82 Token: /, Tipo: DIVISION
83 Token: (, Tipo: PARENTESIS_IZQ
84 Token: g, Tipo: IDENTIFICADOR
85 Token: +, Tipo: SUMA
86 Token: f, Tipo: IDENTIFICADOR
87 Token: ), Tipo: PARENTESIS_DER
88 Token: ,, Tipo: PUNTO_COMA
```

El archivo nos muestra todos los tokens que reconoció el lexer, se encontraron 88 tokens, cada token nos muestra el valor del token y su tipo, el archivo de entrada no contiene errores por lo que se reconoció un token para cada uno de los elementos antes mencionados de la entrada.

Ejemplo 4:

Vamos a probar la siguiente entrada que contiene varios errores:

```
aRboL =42;  
b89012 = a - 7 ;  
c90/ = (a+b) * 2;  
costaricamipatriaquerida = (z+t) * 2;
```

En este ejemplo tenemos 4 líneas de código, las cuales como podemos ver tiene varios errores entre ellos en la primera línea vemos como tenemos mayúsculas y en el lenguaje especificado para este análisis léxico solo se permiten minúsculas, además de que el =42; no contiene espacios como debería entonces también sería un error.

En la segunda línea vemos que el identificador posee números lo cual tampoco se permite.

En la tercera línea el identificador además de poseer números posee un / y más adelante en la línea vemos cómo (a+b) no tiene espacios así como tampoco el 2;

En la última línea el identificador es de más de 12 letras minúsculas, esto no es permitido tampoco según el lenguaje dado y más adelante vemos como el (z+t) y el 2; no posee espacios y representa otro error.

El resultado del lexer se ve así :

```
Token: =, Tipo: ASIGNACION  
Token: a, Tipo: IDENTIFICADOR  
Token: -, Tipo: RESTA  
Token: 7, Tipo: NUMERO  
Token: ;, Tipo: PUNTO_COMA  
Token: =, Tipo: ASIGNACION  
Token: *, Tipo: MULTIPLICACION  
Token: =, Tipo: ASIGNACION  
Token: *, Tipo: MULTIPLICACION  
Error [Fase Léxica]: La línea 1 contiene un error, lexema no reconocido: aRboL  
Error [Fase Léxica]: La línea 1 contiene un error, lexema no reconocido: =42;  
Error [Fase Léxica]: La línea 2 contiene un error, lexema no reconocido: b89012  
Error [Fase Léxica]: La línea 3 contiene un error, lexema no reconocido: c90/  
Error [Fase Léxica]: La línea 3 contiene un error, lexema no reconocido: (a+b)  
Error [Fase Léxica]: La línea 3 contiene un error, lexema no reconocido: 2;  
Error [Fase Léxica]: La línea 4 contiene un error, lexema no reconocido: costaricamipatriaquerida  
Error [Fase Léxica]: La línea 4 contiene un error, lexema no reconocido: (z+t)  
Error [Fase Léxica]: La línea 4 contiene un error, lexema no reconocido: 2;
```