

**Project I: Scanner**  
**Adrián Torres Hernández**  
**A01173530**  
**tecnológico de Monterrey**

## **1.- Introducción**

### **1.1.- Resumen**

#### **Resumen**

El presente reporte presenta los resultados obtenidos en las fases de desarrollo de software para la primera parte en la creación de un compilador "Scanner" el cual es el analizador léxico. El reporte cuenta con la fase de análisis, la cual se encarga en dar una explicación de los componentes del lenguaje Pascal, el desarrollo del Autómata Finito Determinista con la tabla de transiciones y tabla de Tokens ID. Posterior a la fase de análisis sigue la de diseño que se encarga de dar una aproximación a la implementación con pseudo código de como implementar la parte de análisis. Después sigue la fase de implementación en donde se dará una explicación de lo que se hizo en el código. Por último, se termina con la fase de validación y verificación en donde se mostrarán los casos de pruebas con su respectiva entrada y salida.

### **1.2.- Notación**

<b>1.- Introducción</b>	<b>1</b>
<b>1.1.- Resumen</b>	<b>1</b>
<b>1.2.- Notación</b>	<b>1</b>
<b>2.- Análisis</b>	<b>2</b>
<b>2.1 ¿Por qué usar python?</b>	<b>3</b>
<b>2.2 DFA</b>	<b>4</b>
<b>2.3 Tabla de transiciones</b>	<b>4</b>
<b>2.4 Tabla de tokens ID</b>	<b>5</b>
<b>3.- Diseño</b>	<b>6</b>
<b>4.- Implementación</b>	<b>7</b>
<b>4.1 Función scanner</b>	<b>7</b>
<b>4.2 record_token</b>	<b>7</b>
<b>4.3 record_table</b>	<b>8</b>
<b>4.4 Diccionario</b>	<b>9</b>
<b>4.5 transition_table</b>	<b>10</b>
<b>5.- Verificación y Validación</b>	<b>11</b>
<b>6.- Referencias</b>	<b>28</b>

## 2.- Análisis

Para el siguiente proyecto se desarrolló basado en el lenguaje de programación Pascal, el cual, es un lenguaje tipado, con código dividido en fragmentos con semántica propia (funciones o procedimientos) y con variables que han de ser declaradas paso previo a su uso, impidiéndose la interpretación o conversión de valores.[2]

El lenguaje cuenta con los siguientes componentes léxicos:

### Palabras clave (Keywords):

**program:** Se utiliza para iniciar la definición de un programa completo.

**procedure:** Se utiliza para definir un procedimiento sin un valor de retorno.

**function:** Se utiliza para definir una función que devuelve un valor después de su ejecución.

**begin:** Marca el inicio de un bloque de código.

**end:** Marca el final de un bloque de código.

**var:** Se utiliza para declarar variables.

**integer:** Tipo de datos que representa números enteros.

**real:** Tipo de datos que representa números de punto flotante.

**string:** Tipo de datos que representa cadenas de caracteres.

**array:** Se utiliza para declarar arreglos.

**of:** Se utiliza para especificar el tipo de elementos de un arreglo.

**if:** Se utiliza para realizar una condición y ejecutar un bloque de código si esa condición se cumple.

**then:** Se utiliza junto con if para especificar qué hacer si la condición se cumple

**else:** Se utiliza después del para especificar qué hacer si la condición con if no se cumple.

**repeat:** Se utiliza para crear un bucle que se ejecuta repetidamente hasta que se cumple una condición específica.

**until:** Se utiliza con repeat para especificar la condición de salida del bucle.

**for:** Se utiliza para crear un bucle que se ejecuta un número específico de veces.

**to:** Se utiliza junto con for para especificar el límite del rango del bucle.

**do:** Se utiliza junto con for para especificar qué hacer en cada iteración del bucle.

**readLn:** Se utiliza para leer datos desde la entrada y asignarlos a una variable.

**writeLn:** Se utiliza para imprimir datos en la salida.

### Símbolos:

Operación de suma aritmética: +

Operación de resta aritmética: -

Operación de multiplicación aritmética: \*

Operación de división aritmética: /

Operador lógico menor que: <

Operador lógico menor o igual que: <=

Operador lógico mayor que: >

Operador lógico mayor o igual que: >=

Operador lógico igual: =

Operador lógico diferente: <>

Asignación: :=

Punto y coma: ;  
Coma: ,  
Comilla simple: '  
Punto: .  
Paréntesis de apertura: (  
Paréntesis de cierre: )  
Corchete de apertura: [  
Corchete de cierre: ]  
Llave de apertura: {  
Llave de cierre: }  
Comentario de apertura: (\*  
Comentario de cierre: \*)

### **Identificadores:**

Están compuestos por letras, dígitos y guiones bajos ( \_ ) algunos aspectos importantes a destacar son:

- Tienen que iniciar con letra o guiones bajos.
- Si inicia con guion bajo una letra o dígito tiene que ir después.

Ejemplo de identificadores válidos:

num1  
\_num2  
A3br

### **Números:**

Existen dos tipos de números enteros y reales

Ejemplo de número entero: 425

Ejemplo de número real: 425.56

### **Comentarios:**

Los comentarios de una sola línea comienzan con { y cierran con } , y los comentarios de varias líneas están delimitados por (\* y \*)

## **2.1 ¿Por qué usar python?**

Para el siguiente proyecto, se tomó la decisión de desarrollarlo en python debido a su flexibilidad y manejo dinámico de datos para utilizar diccionarios sin mencionar la gran documentación que la respalda, la cual proporciona información útil a la hora de implementar el software. [3]

## 2.2 DFA:

Para el desarrollo del software se diseñó el siguiente autómata de estados finitos se consideraron las restricciones de los identificadores, comentarios, símbolos y estados de error.

[https://drive.google.com/file/d/1qwijzer\\_XG5f4rH7owMIIRxaTM73e53Z/view?usp=sharing](https://drive.google.com/file/d/1qwijzer_XG5f4rH7owMIIRxaTM73e53Z/view?usp=sharing)

Posteriormente, se creó la siguiente traba de transiciones con base en el diagrama de estados finitos.

### 2.3 Tabla de transiciones:

1	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	
2		Letter	Digit	_	.	(	*	)	<	=	>	:	+	-	/	;	,	comilla	.	[	]	{	}	\n	\r	Blank	Other symbols	
3	0	1	4	2	34	7	28	35	10	30	10	12	26	27	29	31	32	13	34	36	37	14	41	0	0	0	42	
4	1	1	1	1	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	42	
5	2	3	3	42	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	42	
6	3	1	1	1	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	42	
7	4	16	4	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	5	16	16	16	16	16	16	16	42	
8	5	17	6	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	42	
9	6	17	6	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	42	
10	7	18	18	18	18	18	8	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	42	
11	8	8	8	8	8	8	9	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	
12	9	8	8	8	8	8	8	19	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	
13	10	20	20	20	20	20	20	20	20	20	21	22	20	20	20	20	20	20	20	20	20	20	20	20	20	20	42	
14	11	23	23	23	23	23	23	23	23	23	24	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	42	
15	12	39	39	39	39	39	39	39	39	25	39	39	39	39	39	39	39	39	39	39	39	39	39	39	39	39	42	
16	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	33	13	13	13	13	13	13	13	13	13	
17	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	40	38	40	40	14	14

Estados aceptores:

[illegible]

Estados de error:

[illegible]

## 2.4 Tabla de tokens ID:

ID	Token
----	-------

18	(
20	<
21	<=
22	<>
23	>
24	>=
25	:=
26	+
27	-
28	*
29	/
30	=
31	;
32	,
33	'
34	.
35	)
36	[
37	]
44	program
45	procedure
46	function
47	begin
48	end
49	var
50	integer
51	real
52	string
53	array

54	of
55	if
56	then
57	else
58	repeat
59	until
60	for
61	to
62	do
63	readLn
64	writeLn

### 3.- Diseño

El pseudocódigo genérico para procesar una tabla de transiciones (Figura 3.1) fue utilizado como base del desarrollo del código debido a la cantidad de estados que hay que analizar en el software. [1]

```

ch = next_input_character();
while (!EOF) {
    state = 0; /* start DFA */
    while (!Accept[state] && !Error[state] ) {
        state = T[state, ch];
        if (Advance[state, ch])          /* See if scanner can get new char */
            ch = next_input_character(); /* given current state and char. */
    } /* end of DFA */
    if (Accept[state] )
        record_Token();
    else
        error_message();
} /* end of FILE */

```

Figura 3.1

### 4.- Implementación

#### 4.1 Función scanner

La función "Scanner" se basa en el pseudocódigo visto en clases, pero en Python, su tarea es abrir el archivo y leer carácter por carácter hasta que se haya acabado. Inicializa el estado en cero y dependiendo del carácter se realizarán diferentes acciones si está en un estado de transición leemos hasta llegar a un estado de éxito. Si encontramos un estado de error mostramos el mensaje de error respectivo al estado. Cuando se lleva a un estado aceptor es necesario enviar el estado aceptor, la cadena completa y el carácter actual con propósito de no perder el ID cuando una cadena llega a un estado aceptor a una función que se encargara de ubicar la cadena a su correspondiente tabla. (Figura 4.1 y 4.2)

```

48 def scanner():
49     with open('archivo.txt') as file:
50         while True:
51             # leer un archivo
52             ch = file.read(1)
53
54             if not ch: #Valida si el archivo esta vacio
55                 break
56
57             state =0 #Iniciamos el DFA
58             word="" #Iniciamos la cadena de identificadores con vacio
59
60             # Bucle para recorrer la tabla de transición hasta que no esté en ninguno de los arreglos de aceptor
61             while state not in Accept and state not in Error:
62                 #print(state, ch)
63                 state=transition_table(state, ch)
64                 if state in Advance: #Valida si es un estado de avance para seguir leyendo
65                     #print(f"está en el arreglo Advance.")
66                     word=word+ch
67                     word=re.sub(r'\s', '', word)
68                     #print(word)
69                     ch = file.read(1)
70                     if not ch: #Valida si el archivo esta vacio
71                         if(state== 8 or state==9):
72                             error_message(state)
73
74                             break
75
76                 #print(state, ch)
77                 word=complete_string(state,word)

```

Figura 4.1

```

478
479     #print("Estado ",state,"letra ",ch)
480     if state in Accept and state not in Comment: # Verificar si el estado está en el arreglo aceptor
481         #print("Estado ",state,"letra ",ch, "cadena de palabra \n", word)
482         record_token(state,ch,word)
483
484     elif state in Error: # Verificar si el estado está en el arreglo error
485         error_message(state)
486

```

Figura 4.2

## 4.2 record\_token

La función "record\_token" separa la cadena dependiendo si su estado aceptor es un identificador, número real o entero de ser el caso, enviamos el estado, la letra y la cadena a una función llamada "record\_table" que se encargara de llenar los

diccionarios de las cadenas y recuperar el símbolo que viene después de la cadena. Si es otro tipo de estado y el carácter no es vacío mostramos el estado aceptor.

```
433
434
435 #Funcion que se encarga de mostrar el estado segun su transicion
436 def record_token (state,ch,word):
437     if(state==15 or state==16 or state==17): #En dado caso que sea un estado de los aceptores de identifica
438         record_table (state,ch,word) #Hay que recuperar el ultimo caracter
439     else:
440         if state is not None and ch not in [' ', '\n', '\r']:
441             print("\n<",state,">")
442
443
```

### 4.3 record\_table

La función "record\_table" sirve para clasificar la cadena ya sea buscar si es un número entero o real, o si es una palabra reservada o un identificador. Evalúa cada caso dependiendo del if si por alguna razón no es palabra reservada revisa que no esté en el diccionario de enteros, reales o identificadores y de no ser así lo agrega con un ID incrementable. Al final, recuperamos el carácter que seguía después dependiendo de su valor para imprimir su id. (Figura 4.3 y 4.4)

```
Scanner.py > record_table
392 #Funcion que sirve para definir si la cadena de los estados
393 def record_table (state,ch,word):
394     #print(state)
395     if check_keyword(word): #Valida si esta en el diccionario de palabras claves
396         idk=find_id(Keywords,word)
397         print("\n<",idk,">")
398
399     elif is_integer(word): #Valida si la cadena es un numero entero de ser asi lo guardamos en la t
400         num = int(word)
401         if num not in Integers.values():
402             # Encontrar el próximo ID disponible
403             idi =max(Integers.keys(), default=0) + 1 #Incrementamos los IDs
404             Integers[idi] = num
405             print("\n<",state,"",idi,">") #Mostramos estado y ID
406         elif num in Integers.values(): #Si el numero ya existe no incrementamos la cadena
407             idi=find_id(Integers,num)
408             print("\n<",state,"",idi,">")
409
410     elif is_float(word): #Valida si la cadena es un numero reales de ser asi lo guardamos en la tab
411         num = float(word)
412         if num not in Reals.values():
413             idf=len(Reals) + 1 #Incrementamos los IDs
414             Reals[idf] = num
415             print("\n<",state,"",idf,">") #Mostramos estado y ID
416         elif num in Reals.values(): #Si el numero ya existe no incrementamos la cadena
417             idf=find_id(Reals,num)
418             print("\n<",state,"",idf,">")
419
```

Figura 4.3



```

419
420 elif not check_keyword(word) and not check_operators(word): #Valida si la cadena no es una palabra r
421     if not check_identifier(word):
422         new_id = len(Identifier) + 1 #Incrementamos los IDs en el diccionario de identificadores
423         Identifier[new_id] = word
424         print("\n<",state,"",new_id,">") #Mostramos estado y ID
425     elif check_identifier(word): #Si la palabra ya existe no incrementamos la cadena
426         new_id=find_id(Identifier,word)
427         print("\n<",state,"",new_id,">")
428
429 if ch is not None and ch not in [' ', '\n', '\r']: # Si la letra quee sigue despues de la cadena es
430     state = find_id(Operators, ch)
431     if state is not None and ch not in [' ', '\n', '\r']:
432         print("\n<", state, ">")
433
434

```

Figura 4.4

## 4.4 Diccionario

Se definió un diccionario de operadores para encontrar el ID acorde al carácter analizado si este sigue después de una cadena. (Figura 4.5)

```

24 # Definir un diccionario con todos los operadores
25 Operators = {
26     18: '(',
27     20: '<',
28     21: '<=',
29     22: '<>',
30     23: '>',
31     24: '>=',
32     25: ':=',
33     26: '+',
34     27: '-',
35     28: '*',
36     29: '/',
37     30: '=',
38     31: ';',
39     32: ',',
40     33: '"',
41     34: '.',
42     35: ')',
43     36: '[',
44     38: ']'
45 }
46

```

Figura 4.5

Se definió un diccionario de palabras clave para encontrar el ID acorde a la cadena recibida. (Figura 4.6)

```

49  # Definir un diccionario con todas las palabras claves
50  Keywords = {
51      44: 'program',
52      45: 'procedure',
53      46: 'function',
54      47: 'begin',
55      48: 'end',
56      49: 'var',
57      50: 'integer',
58      51: 'real',
59      52: 'string',
60      53: 'array',
61      54: 'of',
62      55: 'if',
63      56: 'then',
64      57: 'else',
65      58: 'repeat',
66      59: 'until',
67      60: 'for',
68      61: 'to',
69      62: 'do',
70      63: 'readLn',
71      64: 'writeLn'
72  }

```

Figura 4.6

#### 4.5 transition\_table

Las transiciones en la tabla fueron implementadas con una función llena de condiciones para cada estado en específico y dependiendo del carácter y el estado se retornará un diferente estado. Se tomó la decisión de implementarlo de esa manera porque en un inicio se consideró implementarlo con un diccionario para cada estado, pero no retornaba lo esperado y había casos específicos donde se necesitaba tomar en cuenta los espacios, saltos de línea y tabuladores por lo que con las condiciones en Python puedes especificar esos casos específicos. (Figura 4.7)

```

Scanner.py > record_table
130 #Funcion de tabla de transicion que retorna el estado dependiendo del estado actual y la letra actual
131 #Aqui ocurre la transicion
132 def transition_table(state, ch):
133     if(state < 7):
134         if(state== 0):
135             if ch.isspace():
136                 state=0
137                 return state
138             elif ch.isalpha(): #inicio estado cero
139                 state=1 #inicio de estados no aceptores
140                 return state
141             elif ch== '_':
142                 state=2
143                 return state
144             elif ch.isdigit():
145                 state=4
146                 return state
147             elif ch== '(':
148                 state=7
149                 return state
150             elif ch== '<':
151                 state=10
152                 return state
153             elif ch== '>':
154                 state=11
155                 return state
156             elif ch== ':':
157                 state=12
158                 return state
159             elif ch== '"':
160                 state=13
161                 return state

```

Figura 4.7

## 5.- Verificación y Validación

### Casos de prueba

#### Ejemplo 1:

Entrada:

```

archivo.txt
1  { Example #1 }
2  { This is the typical "Hello World" }
3
4  program HelloWorld;
5
6  (* This is the main program block *)
7  begin
8      writeln( ' Hello World ' );
9  end. (* This is the end of the main
10 program block *)

```

Salida:

```
PS C:\Compiladores> py scanner.py
```

```
< 43 >
```

```
< 15 , 1 >
```

```
< 31 >
```

```
< 46 >
```

```
< 63 >
```

```
< 18 >
```

```
< 33 >
```

```
< 35 >
```

```
< 31 >
```

```
< 47 >
```

```
< 34 >
```

```
Tabla de identificadores
```

```
1 : HelloWorld
```

```
Tabla de numeros enteros
```

```
Tabla de numeros reales
```

```
PS C:\Compiladores>
```

## Ejemplo 2:

Entrada:

```

archivo.txt
1  { Example #2 }
2
3  program Ejemplo2;
4
5  var
6  a : integer;
7  b : real;
8
9  (* This is a procedure block*)
10 procedure assign (x: integer; y: real);
11 begin
12     a := x;
13     b := y;
14 end;
15
16 (* This is the main program block *)
17 begin
18     assign(27, 3.1416);
19     writeln( ' a = ', a );
20     writeln( ' b = ', b );
21 end. (* This is the end of the main
22 program block *)

```

Salida:

```

PS C:\Compiladores> py scanner.py

< 43 >

< 15 , 1 >

< 31 >

< 48 >

< 15 , 2 >

Simbolo de asignación := mal asignado.
PS C:\Compiladores>

```

### Ejemplo 3:

Entrada:

```
archivo.txt
2
3  program Ejemplo2;
4
5  var
6  a := integer;
7  b := real;
8
9  (* This is a procedure block*)
10 procedure assign( x := integer; y := real);
11 begin
12     a := x;
13     b := y;
14 end;
15
16 (* This is the main program block *)
17 begin
18     assign( 27, 3.1416);
19     writeln( ' a = ', a );
20     writeln( ' b = ', b );
21 end. (* This is the end of the main
22 program block *)
```

Salida:

< 43 >  
< 15 , 1 >  
< 31 >  
< 48 >  
< 15 , 2 >  
< 25 >  
< 49 >  
< 31 >  
< 15 , 3 >  
< 25 >  
< 50 >  
< 31 >  
< 44 >  
< 15 , 4 >  
< 18 >  
< 15 , 5 >  
< 25 >  
< 49 >

< 49 >

< 31 >

< 15 , 6 >

< 25 >

< 50 >

< 35 >

< 31 >

< 46 >

< 15 , 2 >

< 25 >

< 15 , 5 >

< 31 >

< 15 , 3 >

< 25 >

< 15 , 6 >

< 31 >

< 47 >

< 31 >

< 46 >



PROBLEMS	OUTPUT	T
< 15 , 4 >		
< 18 >		
< 16 , 1 >		
< 32 >		
< 17 , 1 >		
< 35 >		
< 31 >		
< 63 >		
< 18 >		
< 33 >		
< 32 >		
< 15 , 2 >		
< 35 >		
< 31 >		
< 63 >		
< 18 >		
< 33 >		
< 32 >		
< 15 , 3 >		

< 15 , 3 >

< 35 >

< 31 >

< 47 >

< 34 >

Tabla de identificadores

1 : Ejemplo2

2 : a

3 : b

4 : assign

5 : x

6 : y

Tabla de numeros enteros

1 : 27

Tabla de numeros reales

1 : 3.1416

#### **Ejemplo 4:**

Entrada:

```

archivo.txt
1  { Example #3 }
2
3  program Ejemplo3;
4
5  (* Var declaration section*)
6  var
7      a, b := integer;
8      x, y := real;
9      n := array[ 1 .. 10] of integer;
10     s := string;
11  function calc( w, z := real) := integer;
12  begin
13      if( w >= z) then
14          calc := 5
15      else
16          calc := 0;
17  end;
18  procedure arrayInit( w := integer; z := real );
19  begin
20      for i := 1 to 10 do
21          begin
22              n[ i] := 1 * 5;
23              writeln( 'n[', i, ']', '=', n[ i] );
24          end;
25  end;
26  procedure assign( w, z := real);
27  var
28      temp := real;
29  begin
30      temp := w;
31      repeat
32          temp := temp - z;

```

```
archivo.txt
29 begin
30   temp := w;
31   repeat
32     temp := temp - z;
33   until( temp <= 0);
34   if( temp = 0) then
35     begin
36       a := 10;
37       b := 20;
38     end
39   else
40     begin
41       a := 0;
42       b := 0;
43     end;
44   end;
45   begin
46     s := 'The end';
47     writeln( ' x = ' );
48     readln( x);
49     writeln( ' y = ' );
50     readln( y);
51     if( calc(x,y) := 5) then
52       assign(x,y)
53     else
54       writeln( s);
55   end.
```

Salida:

< 43 >

< 15 , 1 >

< 31 >

< 48 >

< 15 , 2 >

< 32 >

< 15 , 3 >

< 25 >

< 49 >

< 31 >

< 15 , 4 >

< 32 >

< 15 , 5 >

< 25 >

< 50 >

< 31 >

< 15 , 6 >

< 25 >

< 52 >

< 36 >

< 16 , 1 >

< 34 >

< 34 >

< 16 , 2 >

< 38 >

< 53 >

< 49 >

< 31 >

< 15 , 7 >

< 25 >

< 51 >

< 31 >

< 45 >

< 15 , 8 >

< 18 >

< 15 , 9 >

< 32 >

< 15 , 10 >

< 25 >

< 50 >

< 35 >

< 25 >

< 49 >

< 31 >

< 46 >

< 54 >

< 18 >

< 15 , 9 >

< 24 >

< 15 , 10 >

< 35 >

< 55 >

< 15 , 8 >

< 25 >

< 16 , 3 >

< 56 >

< 15 , 8 >

< 25 >

< 16 , 4 >

< 31 >

< 47 >

< 31 >

< 44 >

< 15 , 11 >

< 18 >

< 15 , 9 >

< 25 >

< 49 >

< 31 >

< 15 , 10 >

< 25 >

< 50 >

< 35 >

< 31 >

< 46 >

< 59 >

< 15 , 12 >

< 25 >

< 16 , 1 >

< 60 >

< 16 , 2 >

< 61 >

< 46 >

< 15 , 6 >

< 36 >

< 15 , 12 >

< 38 >

< 25 >

< 16 , 1 >

< 28 >

< 16 , 3 >

< 31 >

< 63 >

< 18 >

< 33 >

< 32 >

< 15 , 12 >

< 32 >

< 33 >

< 32 >

< 15 , 6 >

< 36 >

< 15 , 12 >

< 38 >

< 35 >

< 31 >

< 47 >

< 31 >

< 47 >

< 31 >

< 44 >

< 15 , 13 >

< 18 >

< 15 , 9 >

< 32 >

< 15 , 10 >

< 25 >

< 50 >

< 35 >

< 31 >

< 48 >

< 15 , 14 >

< 25 >

< 50 >

< 31 >

< 46 >

< 15 , 14 >

< 25 >

< 15 , 9 >

< 31 >

< 57 >

< 15 , 14 >

< 25 >

< 15 , 14 >

< 27 >

< 15 , 10 >

< 31 >

< 58 >

< 18 >

< 15 , 14 >

< 21 >

< 16 , 4 >

< 35 >

< 31 >



< 54 >  
< 18 >  
< 15 , 14 >  
< 30 >  
< 16 , 4 >  
< 35 >  
< 55 >  
< 46 >  
< 15 , 2 >  
< 25 >  
< 16 , 2 >  
< 31 >  
< 15 , 3 >  
< 25 >  
< 16 , 5 >  
< 31 >  
< 47 >  
< 56 >  
< 46 >  
< 15 , 2 >  
< 25 >  
< 16 , 4 >  
< 31 >  
< 15 , 3 >  
< 25 >  
< 16 , 4 >  
< 31 >  
< 47 >  
< 31 >  
< 47 >  
< 31 >  
< 46 >  
< 15 , 7 >

< 25 >  
< 33 >  
< 31 >  
< 63 >  
< 18 >  
< 33 >  
< 35 >  
< 31 >  
< 62 >  
< 18 >  
< 15 , 4 >  
< 35 >  
< 31 >  
< 63 >  
< 18 >  
< 33 >  
< 35 >  
< 31 >  
< 62 >  
< 18 >  
< 15 , 5 >  
< 35 >  
< 31 >  
< 54 >  
< 18 >  
< 15 , 8 >  
< 18 >  
< 15 , 4 >  
< 32 >  
< 15 , 5 >  
< 35 >  
< 25 >  
< 16 , 3 >

< 35 >

< 55 >

< 15 , 13 >

< 18 >

< 15 , 4 >

< 32 >

< 15 , 5 >

< 35 >

< 56 >

< 63 >

< 18 >

< 15 , 7 >

< 35 >

< 31 >

< 47 >

< 34 >

#### Tabla de identificadores

1 : Ejemplo3

2 : a

3 : b

4 : x

5 : y

6 : n

7 : s

8 : calc

9 : w

10 : z

11 : arrayInit

12 : i

13 : assign

14 : temp

#### Tabla de números enteros

1 : 1

2 : 10

3 : 5

4 : 0

5 : 20

#### Tabla de números reales

## 6.- Referencias

- 1.- R. Castelló, Class Lecture, Topic: "Chapter 2 – Lexical Analysis." TC3048, School of Engineering and Science,ITESM, Chihuahua, Chih, April, 2020. 14
- 2,- Museu informática, "El Lenguaje de Programación Pascal", S.F; disponible en:<https://museo.inf.upv.es/pascal/> ; Internet; consultado el 12 de octubre 2023
- 3.- Paloma R, "Las 5 razones por las que todo el mundo quiere aprender Python", 5 de febrero de 2020 ; disponible en: <https://telefonicatech.com/blog/las-5-razones-por-las-que-todo-el-mundo-quiere-aprender-python#:~:text=Python%20es%20un%20lenguaje%20de,por%20tanto%2C%20favorece%20la%20productividad> ;Internet; consultado el 12 de octubre 2023