**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**

**1.- Introducción ------------------------------------------------------------------------------------ 1**

**1.1.- Resumen -------------------------------------------------------------------------------------- 1**

**1.2.- Notación --------------------------------------------------------------------------------------- 1**

**2.- Análisis ------------------------------------------------------------------------------------------- 2**

**2.1 ¿Por qué usar python? --------------------------------------------------------------------- 3**

**2.2 DFA ------------------------------------------------------------------------------------------------ 4**

**2.3 Tabla de transiciones ------------------------------------------------------------------------ 4**

**2.4 Tabla de tokens ID --------------------------------------------------------------------------- 5**

**3.- Diseño --------------------------------------------------------------------------------------------- 6**

**4.- Implementación -------------------------------------------------------------------------------- 7**

**4.1 Función scanner ------------------------------------------------------------------------------ 7**

**4.2 record\_token ---------------------------------------------------------------------------------- 7**

**4.3 record\_table ------------------------------------------------------------------------------------ 8**

**4.4 Diccionario -------------------------------------------------------------------------------------- 9**

**4.5 transition\_table -------------------------------------------------------------------------------- 10**

**5.- Verificación y Validación --------------------------------------------------------------------11**

**6.- Referencias ------------------------------------------------------------------------------------- 28**

**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]

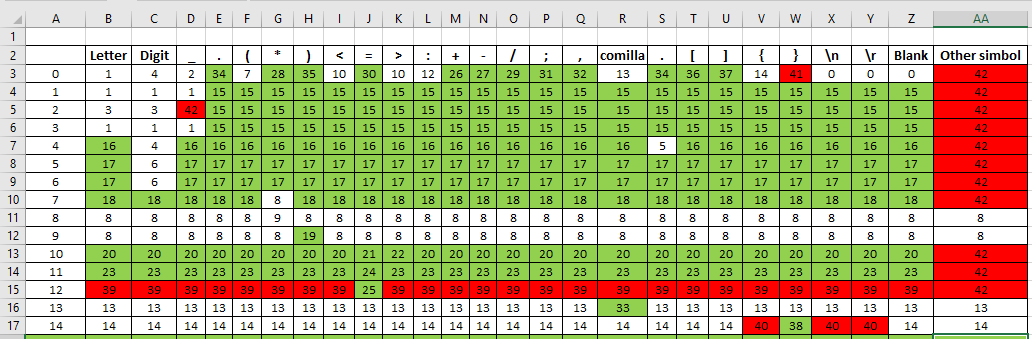
**2.2 DFA**:

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.

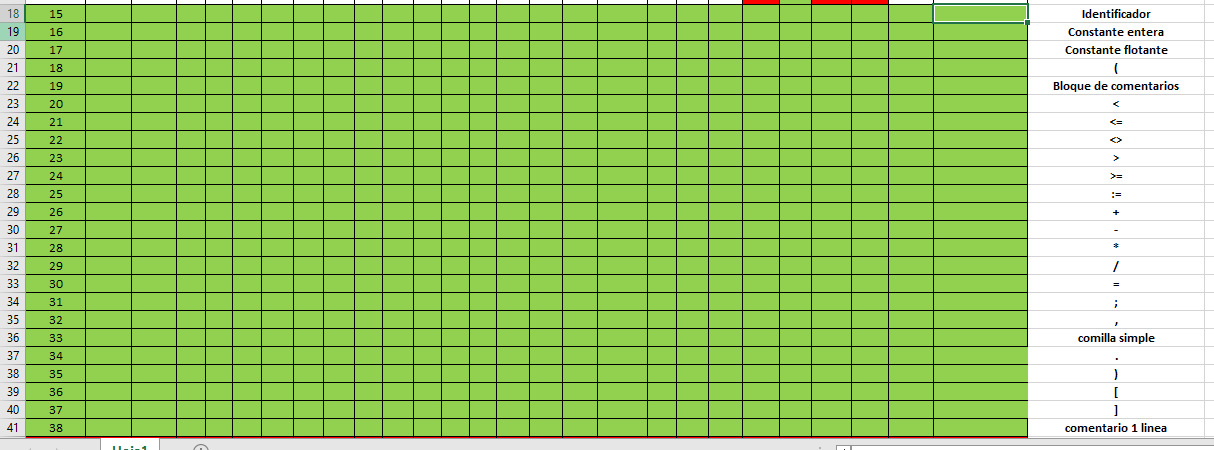
<https://drive.google.com/file/d/1gwjjzer_XG5f4rH7owMIIRxaTM73e53Z/view?usp=sharing>

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

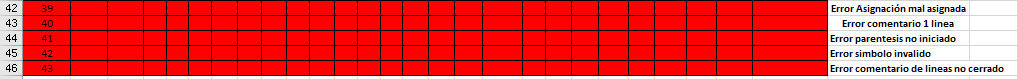
**2.3 Tabla de transiciones:**



Estados aceptores:



Estados de error:



**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]

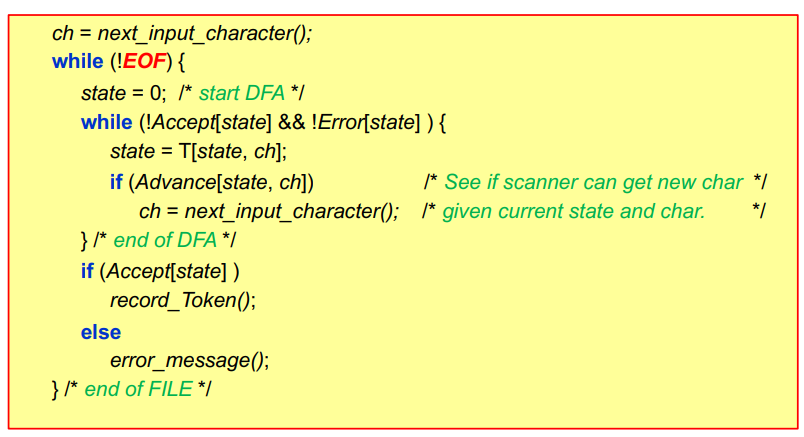
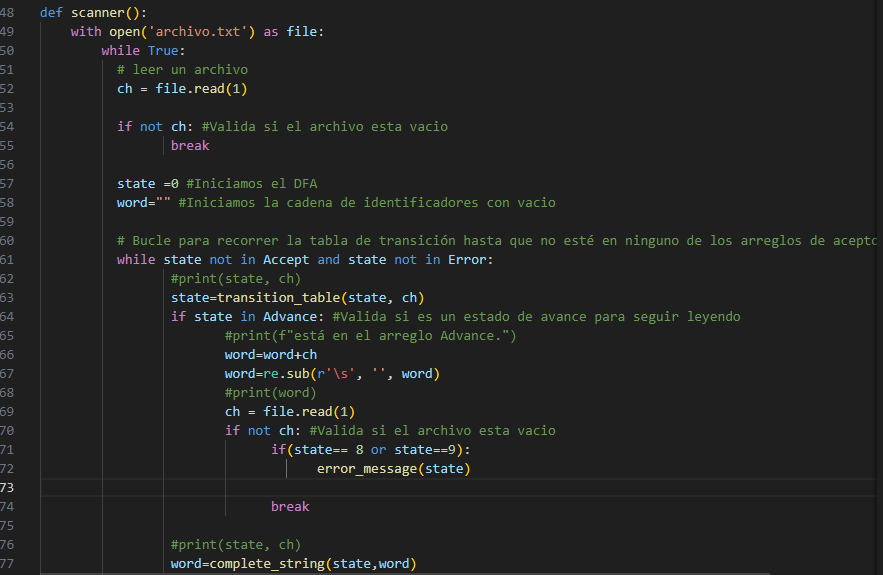


Figura 3.1

**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)

Figura 4.1

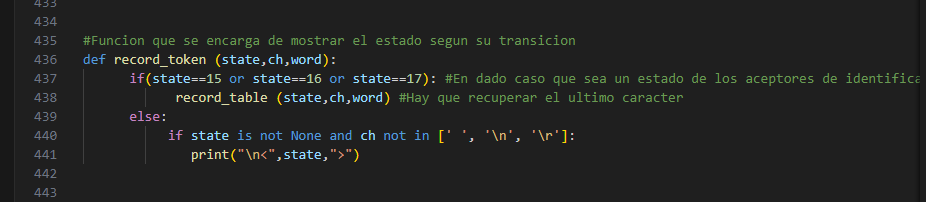
Texto

Descripción generada automáticamente

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.



**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)

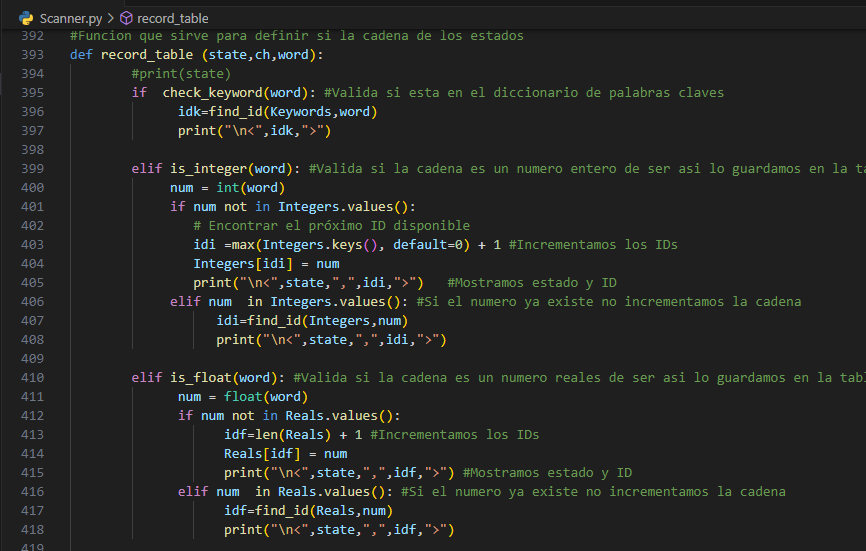


Figura 4.3

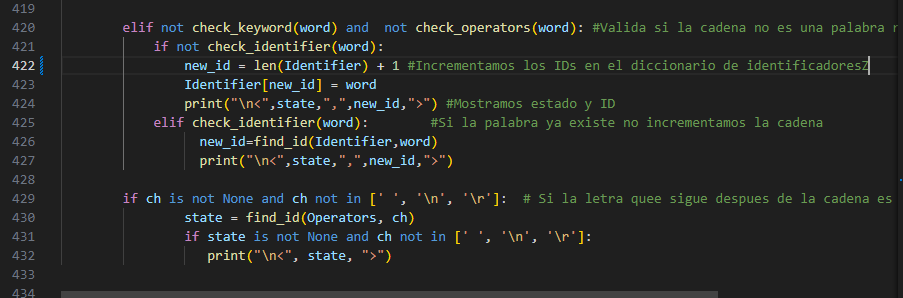


Figura 4.4

**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)

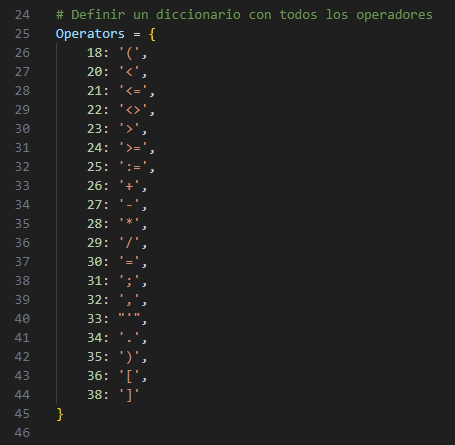


Figura 4.5

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

Texto

Descripción generada automáticamente

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)

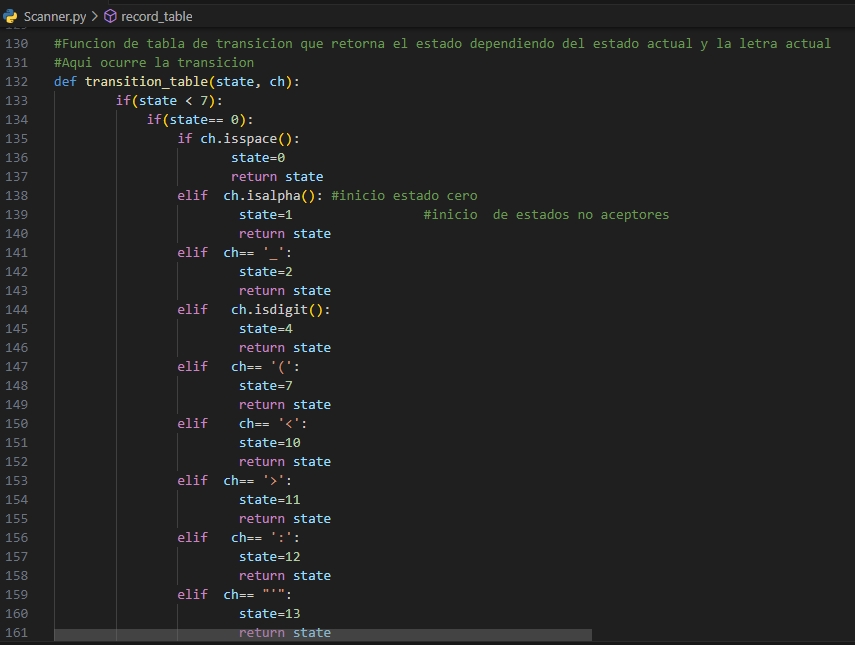


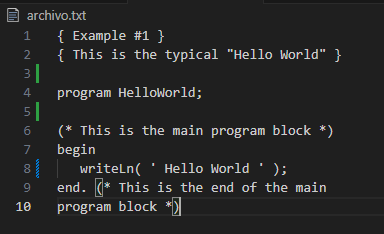
Figura 4.7

**5.- Verificación y Validación**

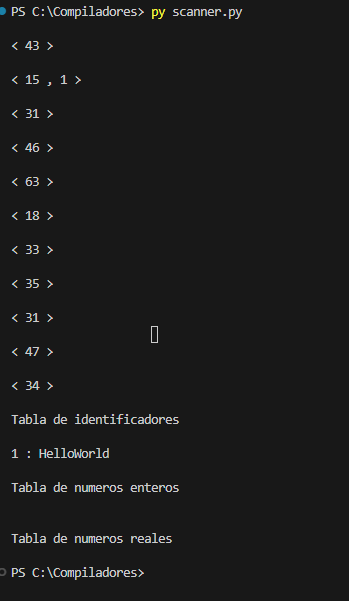
**Casos de prueba**

**Ejemplo 1**:

Entrada:

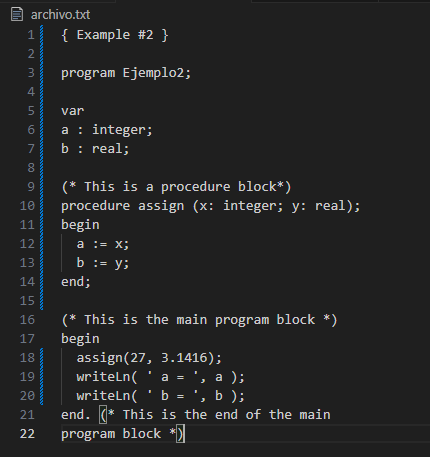


Salida:

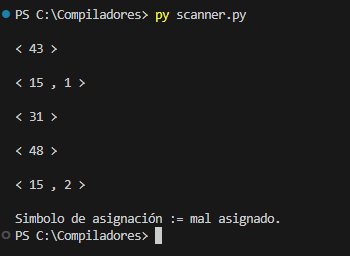


**Ejemplo 2**:

Entrada:

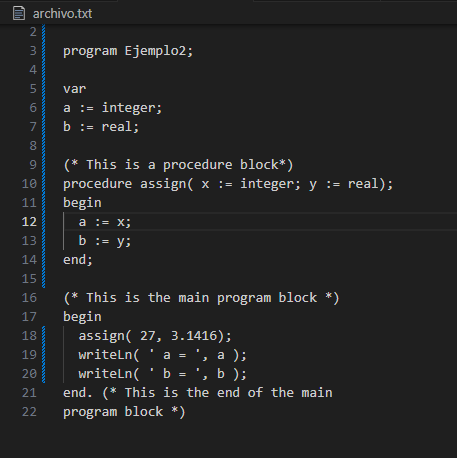


Salida:

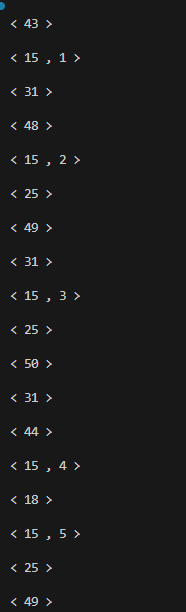


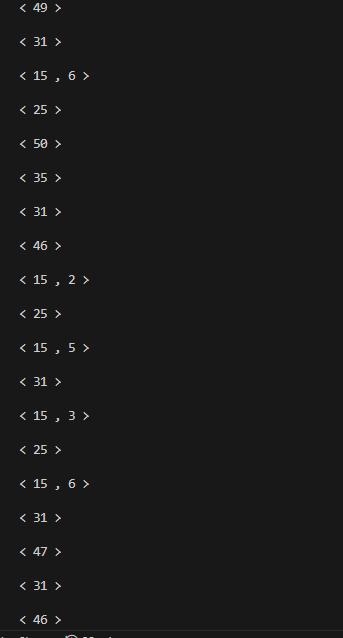
**Ejemplo 3**:

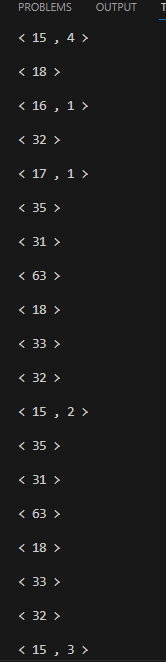
Entrada:

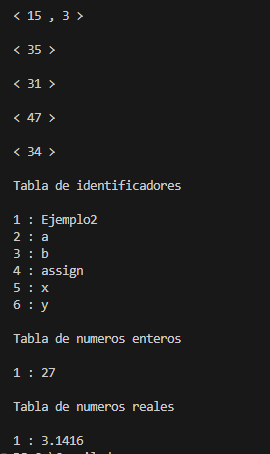


Salida:



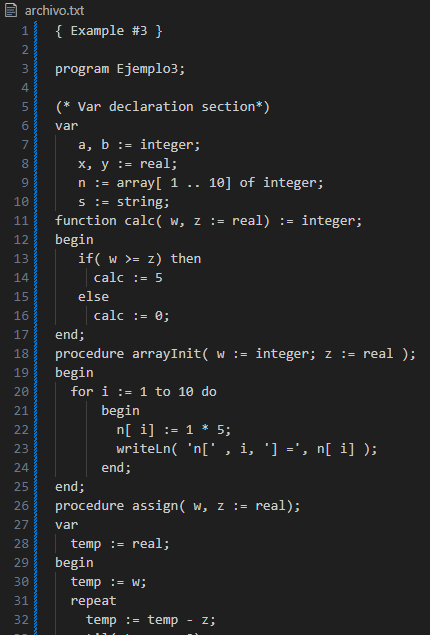


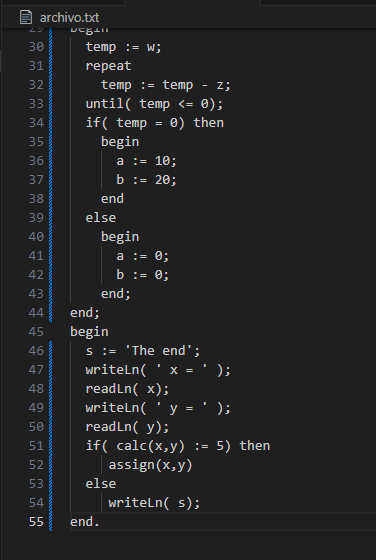




**Ejemplo 4**:

Entrada:





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