UNIVERSIDAD CENTROAMERICANA

JOSÉ SIMEÓN CAÑAS

TEORIA DE LENGUAJES DE PROGRAMACIÓN

FASE 0- ANALIZADOR LEXICOGRÁFICO

GRUPO #10

INTEGRANTES:

OSCAR ALEJANDRO RODRÍGUEZ ABREGO - 00206019

OSCAR AGUSTIN RECINOS ALVARENGA - 00128519

FELIX GERARDO GUEVARA PALACIOS 00055419

IVETTE CAROLINA PINTO LEÓN - 00186319

SEBASTIAN FLORES IRAHETA - 00196919

RODRIGO ALEJANDRO FERNÁNDEZ LEIVA - 00041619

Antiguo Cuscatlán, 06 de octubre de 2023

Se desarrolló un analizador lexicográfico utilizando el lenguaje de programación python en su versión 3.9.13, utilizando la libreria llamada ply la cual es una implementación de lex y yacc las cuales son herramientas muy conocidas para el desarrollo de compiladores, cabe destacar que es necesario definir algunos conceptos de importancia que forman parte de la estructura del analizador lexicográfico tales como:

- Token: Es un par de la forma clave-valor, que consiste en el nombre del token y su respectivo valor de atributo opcional
- Lexema: Se define como una secuencia de caracteres en el programa fuente, que coinciden con el patrón definido para algún token permitiendo al analizador lexico identificar esta secuencia como una instancia de un token.
- Expresión regular: puede definirse como una secuencia de caracteres que define un patrón de búsqueda de acuerdo con ciertas reglas predefinidas.

El analizador lexicográfico desarrollado fue adaptado para leer el lenguaje C# en su ultima versión la cual es la 11.0, el cual es un lenguaje de programación desarrollado por Microsoft que se centra en la simplicidad, la seguridad y la facilidad de uso, se utiliza en una amplia variedad de aplicaciones, desde desarrollo de software de escritorio hasta aplicaciones web y servicios en la nube, y ha evolucionado para admitir el desarrollo multiplataforma en los últimos años.

Uno de los elementos principales utilizados en el analizador lexicográfico son las palabras reservadas, en el presente proyecto se escogieron once palabras reservadas del lenguaje c# las cuales se describen a continuación:

Palabra reservada	**Función**
If	Permite evaluar una condición y, si resulta ser verdadera, ejecuta un bloque de código asociado, pero por el contrario si la condición resulta ser falsa, el bloque de código no se ejecuta.
else	Debe ser anidada siempre a una condició “if”, en caso que el if retorne como resultado falso, el programa ejecutará el bloque de código asociado al "else”
class	Permite definir una clas, a su vez se utiliza para definir un tipo de objeto que puede tener propiedades y métodos (funciones) relacionados.
return	Se utiliza en funciones y métodos para devolver un valor específico al lugar desde donde se llamó, cuando una función detecta una instrucción return, su ejecución se detiene y el valor especificado se devuelve al código que la invocó.

const	Es utilizado para declarar constantes, que son valores que no pueden cambiar después de haber sido asignados.
using	Se utiliza comunmente para espacios de nombres o para la gestión de recursos
int	Se utiliza para definir una variable de tipo entero.
double	Se utiliza para definir una variable de tipo punto flotante, utilizada para representar números de punto flotante en formato de doble precisión.
string	Se utiliza para definir una variable que almacena una secuencia de caracteres, es decir, texto.
bool	Es utilizada para definir una variable de tipo booleano cuyo valor puede ser unicamente true o false.
for	Permiterepetir un bloque de código un número específico de veces.

Posteriormente se definieron los tokens necesarios para que el analizador lexicográfico funcione en base a las reglas principales del lenguaje de programación c#, cabe destacar que fueron definidos bajo el estandar t_nombretoken requerido por la librería ply.lex los cuales se detallan con sus respectivas reglas o patrones definidos de la siguiente manera:

Token	**Expresión regular/regla**	**Explicación**
identificador	**r'[a-zA-Z_][a-zA-Z0-9_]*'**	Permite detectar los identificadores válidos propios del lenguaje,sabiendo que el primer bloque **[a-z A-Z _]** representa el primer carácter, que debe ser una letra (mayúscula o minúscula) o un guión bajo debido a que el lenguaje no permite que un identificador inicie con un número. Por su parte **[a-z A-Z 0-9 _]*** representa los caracteres subsiguientes, que pueden ser letras, dígitos o guiones bajos, en cualquier cantidad.
coma	**r'\,'**	Expresión regular que identifica las comas las cuales son utilizadas comúnmente como separadores en diversas situaciones o expresiones.

comentario	**r'\/\/.*'**	Debido a que el carácter de la barra inclinada **(/)** tiene un significado especial en expresiones regulares, se debe escapar usando una barra inclinada adicional **(\ /)**, por lo tanto la expresión regular **\ / \ / . *** busca todas las cadenas que inicien con dos barras inclinadas y que posteriormente le siga cualquier cantidad de caracteres distintos, al identificar este token, el analizador simplemente lo ignora.
linea_comentario	**r'\/*[\s\S]*?*\/'**	Se utiliza la expresión regular **\ / \ * [\ s \ S] * ? \ * \ /** para buscar cualquier texto encerrado entre **/*** y ***/**. Esto permite reconocer comentarios que pueden abarcar varias líneas, al identificar este patron, el analizador simplemente lo ignora.
fin_de_instruccion	**r'\;'**	Establece el fin de una sentencia o de una instrucción el cual es identificado por un punto y coma
parentesis_de_inicio	**r'\('**	Identifica un parentesis de inicio,que tiene diferentes propositos tales como indicar la llamada o declaración de metodos entre otros
parentesis_de_cierre	**r'\)'**	Identifica un parentesis de cierre
llave_de_inicio	**r'\{'**	Identifica el inicio de un bloque de instrucciones el cual se indica usando una llave izquierda o llave de apertura
llave_de_cierre	**r'\}'**	Identifica una llave derecha o de cierre, la cual indica el fin de un bloque de instrucciones
salto_de_linea	**r'\n+'**	Utilizada para identificar los saltos de lineas presentes en el código de entrada.
ignore	**r' \t'**	Identifica las tabulaciones y espacios presentes en el código de entrada para indicarle al analizador lexicográfico que deben ser ignorados y no deben ser tratados como tokens

		individuales.
punto	**r'\.'**	Identifica los puntos en el código de entrada, usados en c# para llamar a propiedades de objetos, entre otras funciones

Operadores		
Token	**Expresión regular/regla**	**Explicación**
mayor	**r'\>'**	Operadores que son utilizados para realizar comparaciones, mayor que, mayor o igual que, menor que, menor o igual que, igual
mayor_igual_que	**r'\>='**	
menor	**r'\<'**	
menor_igual_que	**r'\<='**	
igual_que	**r'=='**	
mas	**r'\+'**	Operadores para utilizarse en fórmulas dentro de un algoritmo.
menos	**r'\-'**	
multiplicación	**r'*'**	
division	**r'/'**	
distinto	**r'\!='**	Operador que indica que dos elementos son distintos
negacion	**r'\!'**	Indica el valor contrario del que posee un elemento
operador_y	**r'\&\&'**	Operador lógico and
operador_o	**r'|{2}'**	Operador lógico or
modulo	**r'%'**	Operador módulo
restar_acumulado	**r'\-\-'**	Resa 1 a la variable que se le aplica
acumulado	**r'\+\+'**	Suma 1 a la variable que se le aplica
asignación	**r'\='**	Tipo de signo que sirve para almacenar un valor en una variable

A continuación se muestran ejemplos de ejecución del analizador lexicográfico

Ejemplo 1

Código de entrada:

```
1   using System;
2   class Test
3   {
4       //Comentario de linea de prueba
5       static void Main()
6       {
7           /*Comentario de multilinea de
8           prueba*/
9           double num=0.9;
10          string cadena="Prueba";
11          Console.WriteLine(cadena+num);
12      }
13  }
```

Resultado

Al leer el anterior código, se obtiene la siguiente tabla de simbolos

Tabla de simbolos...

Linea	Posicion	Tipo	Valor
1	0	palabra_reservada	using
1	6	identificador	System
1	12	fin_de_instruccion	;
2	14	palabra_reservada	class
2	20	identificador	Test
3	25	llave_de_inicio	{
5	67	identificador	static
5	74	identificador	void
5	79	identificador	Main
5	83	parentesis_de_inicio	(
5	84	parentesis_de_cierre	)
6	90	llave_de_inicio	{
8	155	palabra_reservada	double
8	162	identificador	num
8	165	asignacion	=
8	166	double	0.9
8	169	fin_de_instruccion	;
9	179	palabra_reservada	string
9	186	identificador	cadena
9	192	asignacion	=
9	193	string	"Prueba"
9	201	fin_de_instruccion	;
10	211	identificador	Console
10	218	punto	.
10	219	identificador	WriteLine
10	228	parentesis_de_inicio	(
10	229	identificador	cadena
10	235	mas	+
10	236	identificador	num
10	239	parentesis_de_cierre	)
10	240	fin_de_instruccion	;
11	246	llave_de_cierre	}
12	248	llave_de_cierre	}

Asi mismo se obtiene la siguiente lista de tokens, del archivo “test1.cs”

```
Lista de tokens....


Linea 1      Posicion: 0          Tipo: palabra_reservada       Valor: using
Linea 1      Posicion: 6          Tipo: identificador           Valor: System
Linea 1      Posicion: 12         Tipo: fin_de_instruccion      Valor: ;
Linea 2      Posicion: 14         Tipo: palabra_reservada       Valor: class
Linea 2      Posicion: 20         Tipo: identificador           Valor: Test
Linea 3      Posicion: 25         Tipo: llave_de_inicio         Valor: {
Linea 5      Posicion: 67         Tipo: identificador           Valor: static
Linea 5      Posicion: 74         Tipo: identificador           Valor: void
Linea 5      Posicion: 79         Tipo: identificador           Valor: Main
Linea 5      Posicion: 83         Tipo: parentesis_de_inicio    Valor: (
Linea 5      Posicion: 84         Tipo: parentesis_de_cierre    Valor: )
Linea 6      Posicion: 90         Tipo: llave_de_inicio         Valor: {
Linea 8      Posicion: 155        Tipo: palabra_reservada       Valor: double
Linea 8      Posicion: 162        Tipo: identificador           Valor: num
Linea 8      Posicion: 165        Tipo: asignacion              Valor: =
Linea 8      Posicion: 166        Tipo: double                  Valor: 0.9
Linea 8      Posicion: 169        Tipo: fin_de_instruccion      Valor: ;
Linea 9      Posicion: 179        Tipo: palabra_reservada       Valor: string
Linea 9      Posicion: 186        Tipo: identificador           Valor: cadena
Linea 9      Posicion: 192        Tipo: asignacion              Valor: =
Linea 9      Posicion: 193        Tipo: string                  Valor: "Prueba"
Linea 9      Posicion: 201        Tipo: fin_de_instruccion      Valor: ;
Linea 10     Posicion: 211        Tipo: identificador           Valor: Console
Linea 10     Posicion: 219        Tipo: identificador           Valor: WriteLine
Linea 10     Posicion: 228        Tipo: parentesis_de_inicio    Valor: (
Linea 10     Posicion: 229        Tipo: identificador           Valor: cadena
Linea 10     Posicion: 235        Tipo: mas                     Valor: +
Linea 10     Posicion: 236        Tipo: identificador           Valor: num
Linea 10     Posicion: 239        Tipo: parentesis_de_cierre    Valor: )
Linea 10     Posicion: 240        Tipo: fin_de_instruccion      Valor: ;
Linea 11     Posicion: 246        Tipo: llave_de_cierre         Valor: }
Linea 12     Posicion: 248        Tipo: llave_de_cierre         Valor: }
```

Ejemplo 2

Código de entrada:

```
1  using System;
2  class Prueba1
3  {
4      // Comentario de línea de prueba
5      static void Main()
6      {
7          /* Comentario de múltiples líneas de
8             prueba */
9          int numero = 5;
10         if (numero>=0)
11         {
12             Console.WriteLine("El número es mayor o igual a cero");
13         }
14         else
15         {
16             Console.WriteLine("El número es negativo.");
17         }
18     }
19 }
```

Resultado:

En este ejemplo se cargó el archivo test2.cs que posee un código de ejemplo de c# básico que posee una condición y una salida de texto por consola, luego de su analisis se obtiene la siguiente lista de tokens:

```
Lista de tokens....


Linea 1      Posicion: 0          Tipo: palabra_reservada       Valor: using
Linea 1      Posicion: 6          Tipo: identificador           Valor: System
Linea 1      Posicion: 12         Tipo: fin_de_instruccion      Valor: ;
Linea 2      Posicion: 14         Tipo: palabra_reservada       Valor: class
Linea 2      Posicion: 20         Tipo: identificador           Valor: Prueba1
Linea 3      Posicion: 28         Tipo: llave_de_inicio         Valor: {
Linea 5      Posicion: 71         Tipo: identificador           Valor: static
Linea 5      Posicion: 78         Tipo: identificador           Valor: void
Linea 5      Posicion: 83         Tipo: identificador           Valor: Main
Linea 5      Posicion: 87         Tipo: parentesis_de_inicio    Valor: (
Linea 5      Posicion: 88         Tipo: parentesis_de_cierre    Valor: )
Linea 6      Posicion: 94         Tipo: llave_de_inicio         Valor: {
Linea 8      Posicion: 170        Tipo: palabra_reservada       Valor: int
Linea 8      Posicion: 174        Tipo: identificador           Valor: numero
Linea 8      Posicion: 181        Tipo: asignacion              Valor: =
Linea 8      Posicion: 183        Tipo: int                     Valor: 5
Linea 8      Posicion: 184        Tipo: fin_de_instruccion      Valor: ;
Linea 9      Posicion: 194        Tipo: palabra_reservada       Valor: if
Linea 9      Posicion: 197        Tipo: parentesis_de_inicio    Valor: (
Linea 9      Posicion: 198        Tipo: identificador           Valor: numero
Linea 9      Posicion: 205        Tipo: mayor_igual_que         Valor: >=
Linea 9      Posicion: 208        Tipo: int                     Valor: 0
Linea 9      Posicion: 209        Tipo: parentesis_de_cierre    Valor: )
Linea 10     Posicion: 219        Tipo: llave_de_inicio         Valor: {
Linea 11     Posicion: 233        Tipo: identificador           Valor: Console
Linea 11     Posicion: 240        Tipo: punto                   Valor: .
Linea 11     Posicion: 241        Tipo: identificador           Valor: WriteLine
Linea 11     Posicion: 250        Tipo: parentesis_de_inicio    Valor: (
Linea 11     Posicion: 251        Tipo: string                  Valor: "El número es mayor o igual a cero"
Linea 11     Posicion: 286        Tipo: parentesis_de_cierre    Valor: )
Linea 11     Posicion: 287        Tipo: fin_de_instruccion      Valor: ;
Linea 12     Posicion: 301        Tipo: palabra_reservada       Valor: return
Linea 12     Posicion: 307        Tipo: fin_de_instruccion      Valor: ;
Linea 13     Posicion: 317        Tipo: llave_de_cierre         Valor: }
Linea 14     Posicion: 327        Tipo: identificador           Valor: Console
Linea 14     Posicion: 334        Tipo: punto                   Valor: .
Linea 14     Posicion: 335        Tipo: identificador           Valor: WriteLine
Linea 14     Posicion: 344        Tipo: parentesis_de_inicio    Valor: (
Linea 14     Posicion: 345        Tipo: string                  Valor: "El número es negativo."
Linea 14     Posicion: 369        Tipo: parentesis_de_cierre    Valor: )
Linea 14     Posicion: 370        Tipo: fin_de_instruccion      Valor: ;
Linea 15     Posicion: 376        Tipo: llave_de_cierre         Valor: }
Linea 16     Posicion: 378        Tipo: llave_de_cierre         Valor: }
```

Posteriormente se obtiene la siguiente tabla de simbolos:

Tabla de simbolos...

Linea	Posicion	Tipo	Valor
1	0	palabra_reservada	using
1	6	identificador	System
1	12	fin_de_instruccion	;
2	14	palabra_reservada	class
2	20	identificador	Prueba1
3	28	llave_de_inicio	{
5	71	identificador	static
5	78	identificador	void
5	83	identificador	Main
5	87	parentesis_de_inicio	(
5	88	parentesis_de_cierre	)
6	94	llave_de_inicio	{
8	170	palabra_reservada	int
8	174	identificador	numero
8	181	asignacion	=
8	183	int	5
8	184	fin_de_instruccion	;
9	194	palabra_reservada	if
9	197	parentesis_de_inicio	(
9	198	identificador	numero
9	205	mayor_igual_que	>=
9	208	int	0
9	209	parentesis_de_cierre	)
10	219	llave_de_inicio	{
11	233	identificador	Console
11	240	punto	.
11	241	identificador	WriteLine
11	250	parentesis_de_inicio	(
11	251	string	"El número es mayor o igual a cero"
11	286	parentesis_de_cierre	)
11	287	fin_de_instruccion	;
12	301	palabra_reservada	return
12	307	fin_de_instruccion	;
13	317	llave_de_cierre	}
14	327	identificador	Console
14	334	punto	.
14	335	identificador	WriteLine
14	344	parentesis_de_inicio	(
14	345	string	"El número es negativo."
14	369	parentesis_de_cierre	)
14	370	fin_de_instruccion	;
15	376	llave_de_cierre	}
16	378	llave_de_cierre	}

Ejemplo 3

Código de entrada:

```
using System;
class Cadena
{
    static void Main()
    {
        // Comentario de línea de prueba
        string cadena = "Esto Es Una Prueba";
        bool contieneA = cadena.contains("a"); // Variable booleana
        if (contieneA)
        {
            Console.WriteLine("La cadena contiene 'a'");
            return;
        }
        Console.WriteLine("La cadena no contiene 'a'");
    }
}
```

El anterior código verifica si la variable llamada cadena posee el caracter “a” y posteriormente evalua el valor booleano utilizando una condicion if.

Obteniendo como resultado el siguiente listado de tokens:

```
Linea 1     Posicion: 0          Tipo: palabra_reservada      Valor: using
Linea 1     Posicion: 6          Tipo: identificador          Valor: System
Linea 1     Posicion: 12         Tipo: fin_de_instruccion     Valor: ;
Linea 2     Posicion: 14         Tipo: palabra_reservada      Valor: class
Linea 2     Posicion: 20         Tipo: identificador          Valor: Cadena
Linea 3     Posicion: 27         Tipo: llave_de_inicio        Valor: {
Linea 4     Posicion: 33         Tipo: identificador          Valor: static
Linea 4     Posicion: 40         Tipo: identificador          Valor: void
Linea 4     Posicion: 45         Tipo: identificador          Valor: Main
Linea 4     Posicion: 49         Tipo: parentesis_de_inicio   Valor: (
Linea 4     Posicion: 50         Tipo: parentesis_de_cierre   Valor: )
Linea 5     Posicion: 56         Tipo: llave_de_inicio        Valor: {
Linea 7     Posicion: 107        Tipo: palabra_reservada      Valor: string
Linea 7     Posicion: 114        Tipo: identificador          Valor: cadena
Linea 7     Posicion: 121        Tipo: asignacion             Valor: =
Linea 7     Posicion: 123        Tipo: string                 Valor: "Esto Es Una Prueba"
Linea 7     Posicion: 143        Tipo: fin_de_instruccion     Valor: ;
Linea 8     Posicion: 153        Tipo: palabra_reservada      Valor: bool
Linea 8     Posicion: 158        Tipo: identificador          Valor: contieneA
Linea 8     Posicion: 168        Tipo: asignacion             Valor: =
Linea 8     Posicion: 170        Tipo: identificador          Valor: cadena
Linea 8     Posicion: 176        Tipo: punto                  Valor: .
Linea 8     Posicion: 177        Tipo: identificador          Valor: contains
Linea 8     Posicion: 185        Tipo: parentesis_de_inicio   Valor: (
Linea 8     Posicion: 186        Tipo: string                 Valor: "a"
Linea 8     Posicion: 189        Tipo: parentesis_de_cierre   Valor: )
Linea 8     Posicion: 190        Tipo: fin_de_instruccion     Valor: ;
Linea 9     Posicion: 221        Tipo: palabra_reservada      Valor: if
Linea 9     Posicion: 224        Tipo: parentesis_de_inicio   Valor: (
Linea 9     Posicion: 225        Tipo: identificador          Valor: contieneA
Linea 9     Posicion: 234        Tipo: parentesis_de_cierre   Valor: )
Linea 10    Posicion: 244        Tipo: llave_de_inicio        Valor: {
Linea 11    Posicion: 258        Tipo: identificador          Valor: Console
Linea 11    Posicion: 265        Tipo: punto                  Valor: .
Linea 11    Posicion: 266        Tipo: identificador          Valor: WriteLine
Linea 11    Posicion: 275        Tipo: parentesis_de_inicio   Valor: (
Linea 11    Posicion: 276        Tipo: string                 Valor: "La cadena contiene 'a'"
Linea 11    Posicion: 300        Tipo: parentesis_de_cierre   Valor: )
Linea 11    Posicion: 301        Tipo: fin_de_instruccion     Valor: ;
Linea 12    Posicion: 315        Tipo: palabra_reservada      Valor: return
Linea 12    Posicion: 321        Tipo: fin_de_instruccion     Valor: ;
Linea 13    Posicion: 331        Tipo: llave_de_cierre        Valor: }
Linea 14    Posicion: 341        Tipo: identificador          Valor: Console
Linea 14    Posicion: 348        Tipo: punto                  Valor: .
Linea 14    Posicion: 349        Tipo: identificador          Valor: WriteLine
Linea 14    Posicion: 358        Tipo: parentesis_de_inicio   Valor: (
Linea 14    Posicion: 359        Tipo: string                 Valor: "La cadena no contiene 'a'"
Linea 14    Posicion: 386        Tipo: parentesis_de_cierre   Valor: )
Linea 14    Posicion: 387        Tipo: fin_de_instruccion     Valor: ;
Linea 15    Posicion: 393        Tipo: llave_de_cierre        Valor: }
Linea 16    Posicion: 395        Tipo: llave_de_cierre        Valor: }
```

Y finalmente obteniendo la siguiente tabla de simbolos

Tabla de simbolos...

Linea	Posicion	Tipo	Valor
1	0	palabra_reservada	using
1	6	identificador	System
1	12	fin_de_instruccion	;
2	14	palabra_reservada	class
2	20	identificador	Cadena
3	27	llave_de_inicio	{
4	33	identificador	static
4	40	identificador	void
4	45	identificador	Main
4	49	parentesis_de_inicio	(
4	50	parentesis_de_cierre	)
5	56	llave_de_inicio	{
7	107	palabra_reservada	string
7	114	identificador	cadena
7	121	asignacion	=
7	123	string	"Esto Es Una Prueba"
7	143	fin_de_instruccion	;
8	153	palabra_reservada	bool
8	158	identificador	contieneA
8	168	asignacion	=
8	170	identificador	cadena
8	176	punto	.
8	177	identificador	contains
8	185	parentesis_de_inicio	(
8	186	string	"a"
8	189	parentesis_de_cierre	)
8	190	fin_de_instruccion	;
9	221	palabra_reservada	if
9	224	parentesis_de_inicio	(
9	225	identificador	contieneA
9	234	parentesis_de_cierre	)
10	244	llave_de_inicio	{
11	258	identificador	Console
11	265	punto	.
11	266	identificador	WriteLine
11	275	parentesis_de_inicio	(
11	276	string	"La cadena contiene 'a'"
11	300	parentesis_de_cierre	)
11	301	fin_de_instruccion	;
12	315	palabra_reservada	return
12	321	fin_de_instruccion	;
13	331	llave_de_cierre	}
14	341	identificador	Console
14	348	punto	.
14	349	identificador	WriteLine
14	358	parentesis_de_inicio	(
14	359	string	"La cadena no contiene 'a'"
14	386	parentesis_de_cierre	)
14	387	fin_de_instruccion	;
15	393	llave_de_cierre	}
16	395	llave_de_cierre	}

Información adicional

En la carpeta principal se encontrarán tres archivos ejecutables llamados ejemplo1, ejemplo2 y ejemplo3, estos pueden ser ejecutados y se mostrarán los resultados de la lista de tokens y tabla de hash resultantes de analizar los archivos test1.cs, test2.cs y test3.cs respectivamente que se encuentran en la carpeta llamada test, si se desean hacer pruebas adicionales basta con cambiar parte del código de ejemplo de alguno de estos archivos y abrir el ejecutable que lee el archivo correspondiente de la carpeta test, en caso de detectar comentarios o tokens que no pertenecen al léxico del lenguaje no son mostrados, se ignoran y se sigue la ejecución del programa mostrando unicamente los tokens válidos, además si se desea ejecutar el código fuente basta con ejecutar el archivo analizador.py que se encuentra en la carpeta analizador_lexico y tener importados los dos archivos de la carpeta utils llamados reservadas.py y tokens.py los cuales contienen todos los tokens y palabras reservadas utilizadas junto con los archivos de la carpeta test según se estime conveniente.