



UNIVERSIDAD DE
GUADALAJARA
Red Universitaria e Institución Benemérita de Jalisco



Alumno:

Romo Rodríguez José Alberto (216853747)

Materia:

I7028 | Seminario de Solucion de Problemas de Traductores de Lenguajes II

NRC:

103841

Sección:

Do2

Maestro:

Lopez Franco Michel Emanuel

Horario:

Lunes y Miercoles | 13:00 - 14:55

Tarea:

Mini Generador Lexico

Fecha de Entrega:

23 de Enero del 2022

Indicaciones.

Genera un pequeño analizador léxico, que identifique los siguientes tokens (identificadores y números reales) construidos de la siguiente manera.

identificadores = letra(letra|digito)*

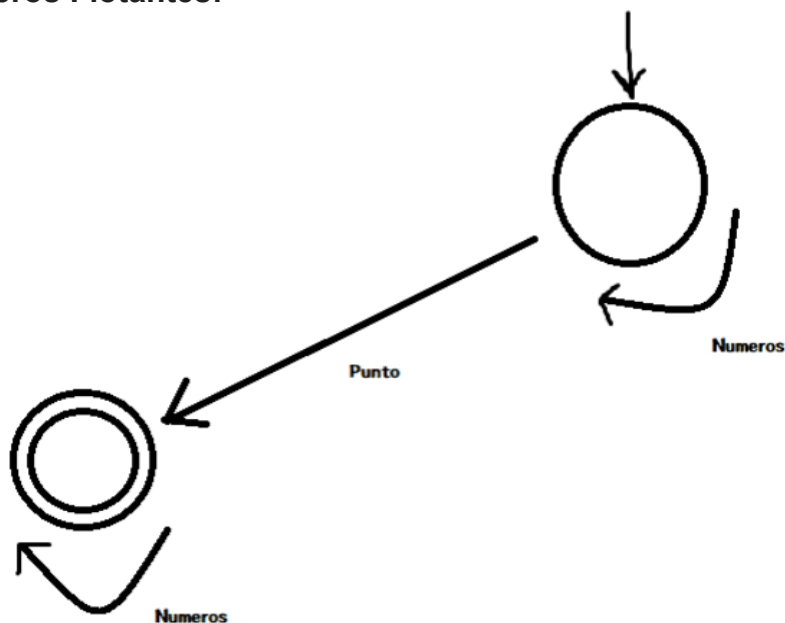
Real = entero.entero+

Introduccion.

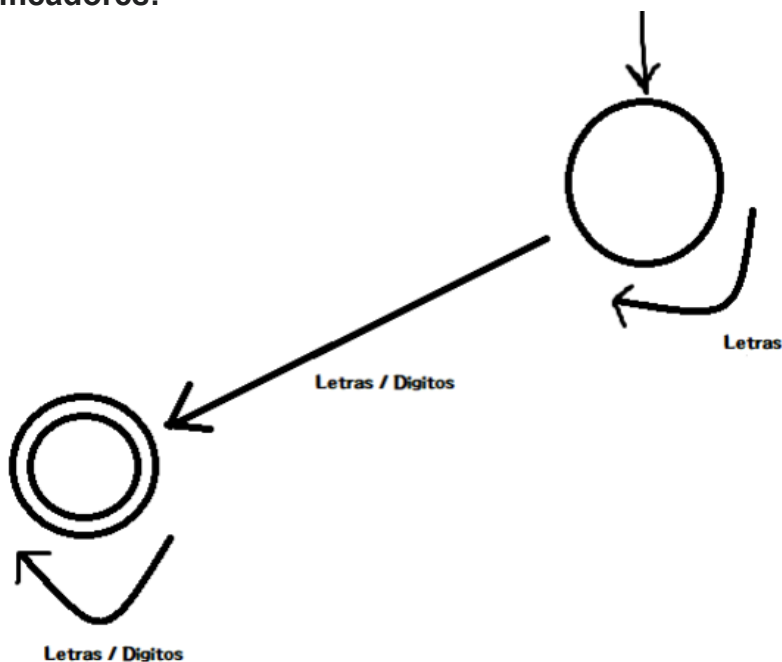
Para comenzar con el desarrollo de este analizador lexico, se realiza un pequeño automata que ayudara a optimizar y abstraer este tipo de cuestiones, pues, de cualquier otro modo, si bien podriamos llegar a la misma solucion, posiblemente fuese algo mas arbitrario y no un metodo como definiriamos correcto para su posterior explicacion.

Asi que establecemos un Automata sencillo que resulta de la siguiente manera:

Numeros Flotantes:

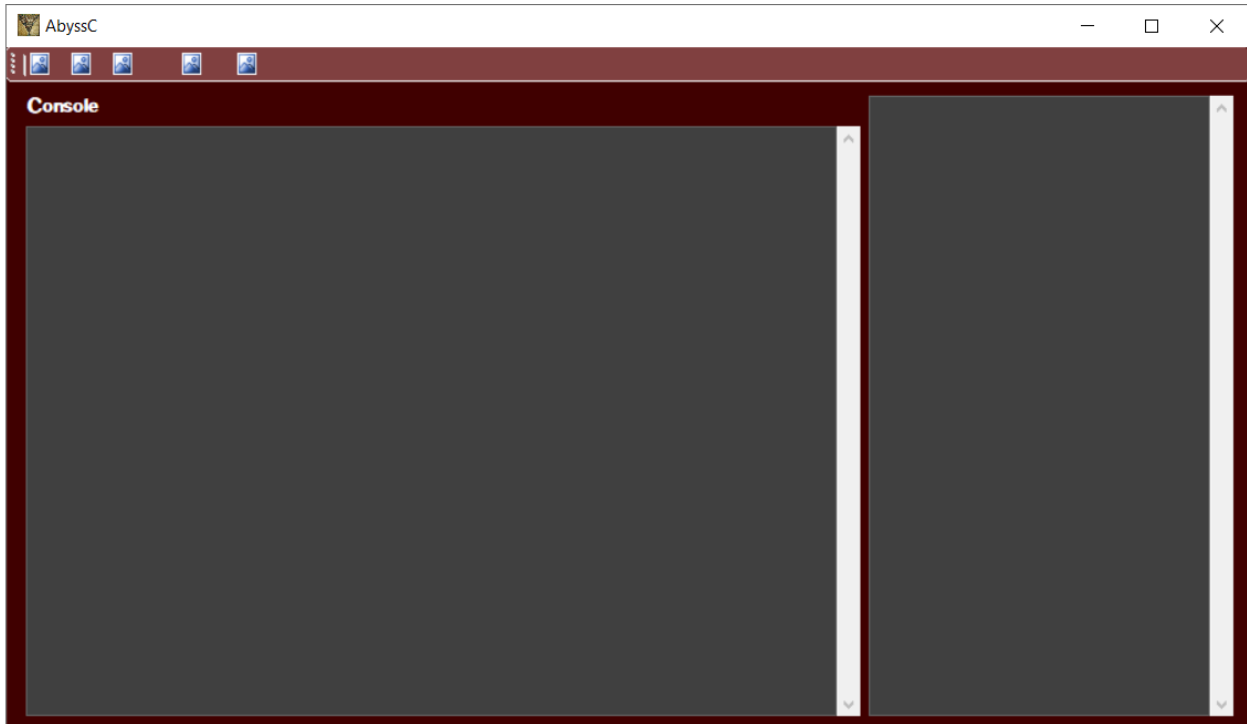


Identificadores:



Como se puede ver son practicamente de la misma forma, entonces cualquier cosa fuera de ese esquema no sera, de manera respectiva, elementos que estemos validando.

En seguida el lenguaje elegido para hacer desarrollo de estos metodos sera C#, especificamente haciendo uso de Windows Form para a su vez realizar una interfaz un poco mas amigable y en la cual poder trabajar posteriormente.



A este proyecto se le dio el nombre de AbyssC, en referencia al gran abismo que aparenta ser c como lenguaje, «Abismo C».

Desarrollo.

Una vez listos los preparativos y con la logica lista solamente toca plasmarlo en C#, de la siguiente manera:

Definimos el alfabeto:

```
Identificador = new string[28] {"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", "_"};
```

Nota*: Se incluye «_» debido a que en los lenguajes que suelo usar _ es permitido como una letra mas dentro de los identificadores.

Definimos los numeros:

```
Entero = new string[10] { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};  
Real = new string[11] { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "."};
```

Una vez definidas nuestros diccionarios hacemos las validaciones, para facilitar las cosas se usara el metodo .Contains<type>(type) que realiza una busqueda dentro de determinado arreglo y nos retorna un booleano que indica si lo contiene o no.

Realizamos el automata:

Primero tomamos el primer valor de la cadena y lo comparamos con numeros reales debido a que si yo introduzco «.05» realmente es correcto y si lo comparo con Enteros el ».

pasaria descartado.

```
if(Real.Contains<string>(word[0].ToString()))
{
    if (word[0] == '.')
    {
        opc = 2;
    }
    else
    {
        opc = 1;
    }
    i++;
}
```

En seguida tras comprobar que estamos tratando con un numero diferenciamos, si el primer valor es un «.» significa que es real, de momento, queda la posterior evaluacion en el automata, en cambio si es un numero cualquiera entra al estado de numero entero y entra al automata de igual manera.

```
switch(opc)
{
    case 1:
        if (Entero.Contains<string>(word[i].ToString()))
        {
            opc = 1;
        }
        else if (word[i] == '.')
        {
            opc = 2;
        }
        else
        {
            opc = -1;
        }
        break;
    case 2:
        if (Entero.Contains<string>(word[i].ToString()))
        {
            opc = 2;
        }
        else
        {
            opc = -1;
        }
        break;
}
```

Segun cual sea el caso entrara en este automata pequeño que simplemente validara mientras se recorre la palabra, esto gracias al uso de un while que se encarga de ir recorriendo la palabra con un contador.

Si el estado en cuestion no esta controlado opc, pasa a ser -1 y al termino de la validacion de la expresion retornamos el resultado:

```

}
if(opc == -1)
{
    return 24;
}
if(opc == 1)
{
    return 1;
}
if(opc == 2)
{
    return 2;
}

```

Realizado de esta manera debido a que al retornar 24 es un caso de Error, 1 es Entero y 2 es Real, y según la posición en arreglo imprime, de la siguiente manera:

El arreglo del que se habló hace un momento:

```

Resultados = new string[25] {"Identificador", "Entero", "Real", "Cadena",
    "Tipo", "OpSuma", "OpMul", "OpRelac", "OpOr", "OpAnd", "OpNot",
    "OpIgualdad", "PuntoComa", "Coma", "ParentesisOpen", "ParentesisClose",
    "CorcheteOpen", "CorcheteClose", "Igual", "_if", "_while", "_return",
    "_else", "_terminal", "ERROR"};

```

El uso de impresión en cuestión:

```

Lexico.Add(word);
Results.Text += word + " -> " + Resultados[x] + "\r\n";

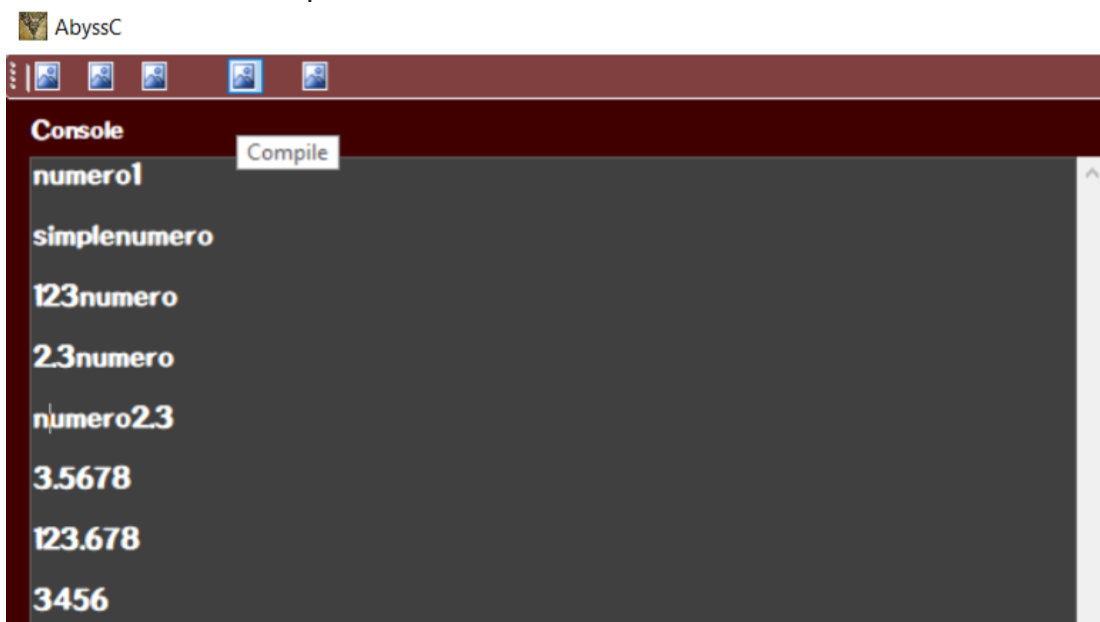
```

De esta manera tenemos una manera un poco más ordenada en que suceden las cosas y de ser necesario podemos cambiar texto de los resultados.

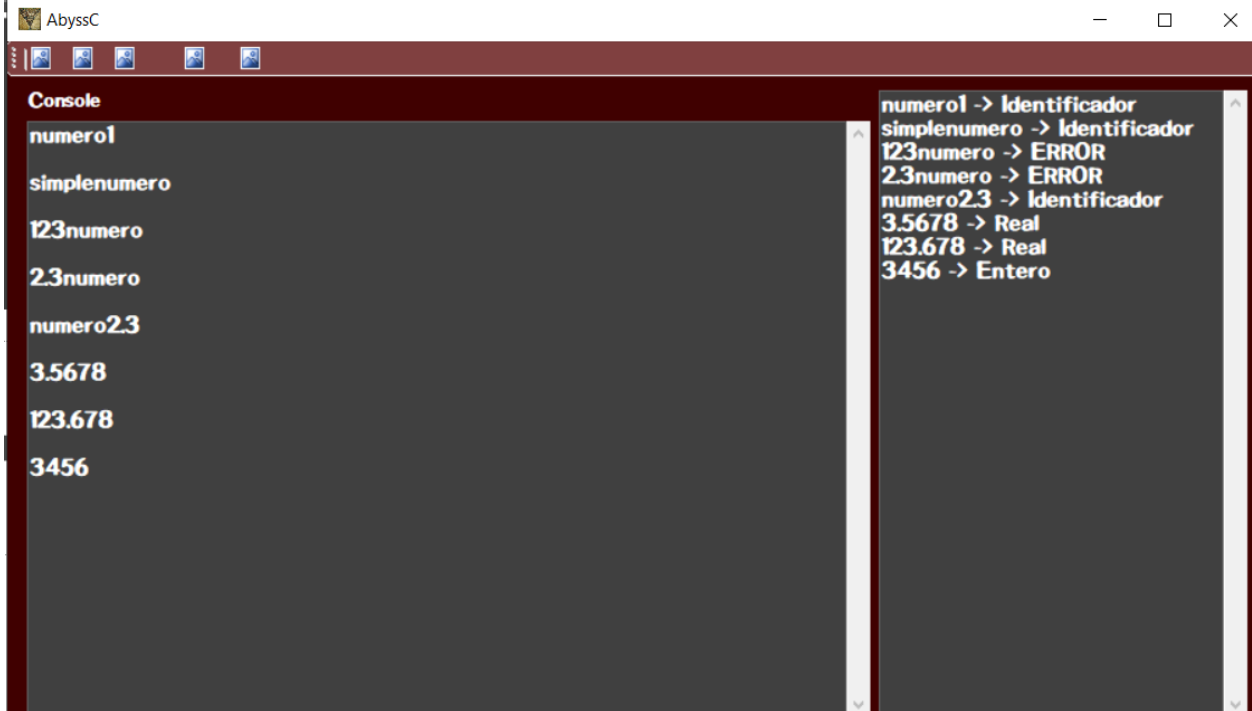
Conclusiones.

Para terminar esta parte simplemente ejecutamos nuestro código y observamos los resultados:

Colocamos nuestras palabras:



Ejecutamos:



The screenshot shows a console window titled 'AbyssC' with a dark background and light text. The window is split into two panes. The left pane, labeled 'Console', displays a list of input strings: 'numero1', 'simplenúmero', '123numero', '2.3numero', 'numero2.3', '3.5678', '123.678', and '3456'. The right pane displays the corresponding tokenization results for each input string, separated by a vertical line. The results are: 'numero1 -> Identificador', 'simplenúmero -> Identificador', '123numero -> ERROR', '2.3numero -> ERROR', 'numero2.3 -> Identificador', '3.5678 -> Real', '123.678 -> Real', and '3456 -> Entero'.

Input	Output
numero1	numero1 -> Identificador
simplenúmero	simplenúmero -> Identificador
123numero	123numero -> ERROR
2.3numero	2.3numero -> ERROR
numero2.3	numero2.3 -> Identificador
3.5678	3.5678 -> Real
123.678	123.678 -> Real
3456	3456 -> Entero

Y podemos apreciar que efectivamente es correcto.