



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:

Analizador Lexico Completo

Fecha de Entrega:

23 de Enero del 2022

Introduccion.

Se realizan diccionarios para cada uno de los tipos de datos establecidos en la actividad, es decir, arreglos con los elementos especificos que pueden o deben contener una palabra en cuestion para ser considerada de algun tipo posible, los tipos a usar son los siguientes:

Identificador - Entero - Real - Cadena - Tipo - OpSuma - OpMul - OpRelac - OpOr - OpAnd - OpNot - OpIgualdad - PuntoComa - Coma - ParentesisOpen - ParentesisClose - CorcheteOpen - CorcheteClose - Igual - _if - _while - _return - _else - _terminal.

Todos estos estan definidos en la tabla de la actividad como simbolos lexicos para ser reconocidos:

Identificadores= letra (letra|digito)*

Entero= digito⁺

Real= entero.entero

Operador de adición: + | -

Operador de multiplicación: * | /

Operador de asignación: =

Operador relacional: < | > | <= | >= | != | ==

Operador And: &&

Operador Or: ||

Operador Not: !

Parentesis: (,)

Llave: { , }

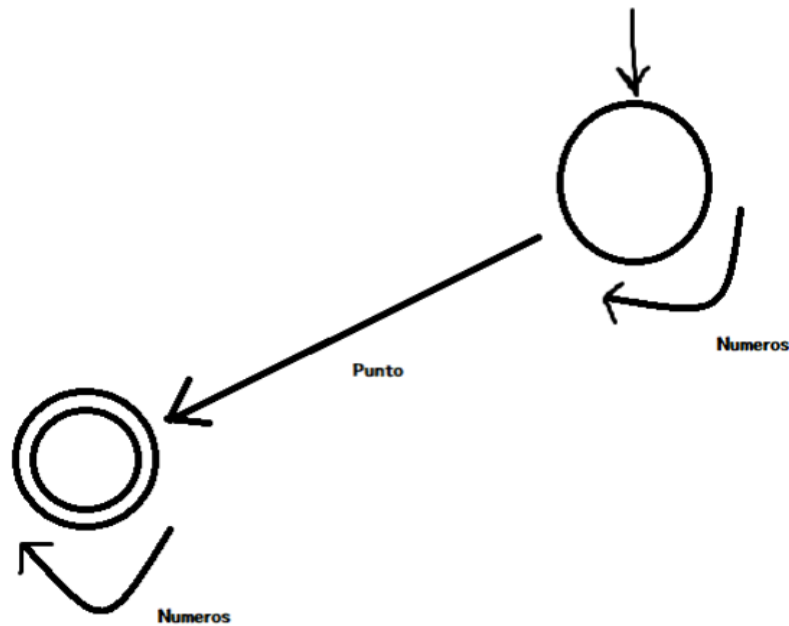
Punto y coma: ;

El entorno en el que se trabajara es C#, siendo precisos en la parte de Windows Form, pues, de esta manera podemos desarrollar una interfaz mas agradable para el usuario y mostrar los resultados se hace mas sencillo.

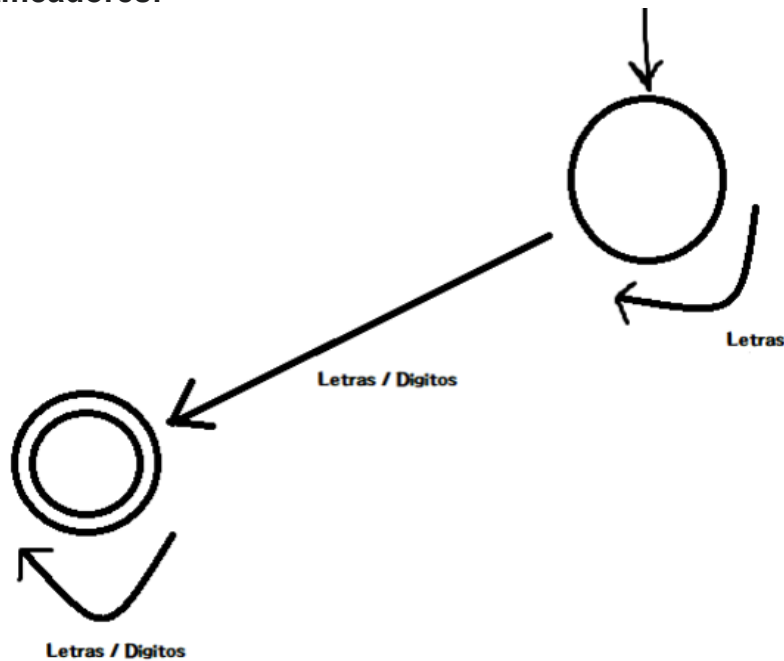
Para realizar las validaciones de Identificadores, Entero y Real se realizaron 2 automatas que se usaron para la actividad anterior, ademas de eso para el resto de validaciones realizamos un sistema de discriminacion que separa las palabras por distintos tipos de separadores comunes y ademas se usan algunos de los mismo simbolos, pues, en muchas ocasiones estos se encuentran unidos a numeros u otros operadores.

En seguida se muestran los automatas:

Numeros Flotantes:



Identificadores:



Ahora, tras tener todo listo pasamos a la parte de codificación.

Desarrollo.

Para comenzar, realizamos la interfaz del programa que nos ayudara a ingresar la informacion y tambien mostrar los resultados para poder comprobar que todo lo que haremos es correcto o incorrecto dependiendo cual sea el caso.

Enseguida, se hacen todos los diccionarios que nos facilitaran la busqueda y comprobacion y ademas se les coloca sus respectivos valores, se hace a manera de arreglo simple, pues, no es necesario incrementar los tamaños de los arreglos en tiempo de ejecucion, aunque posteriormente sera necesario realizar arreglos dinamicos para almacenar infor-

macion.

Interfaz:



Codigo:

```
//Tokens -> Orden 0 a 21 en este orden
public string[] Identificador = new string[28]; //LetrasNumeros < Nada de simbolos solo _
public string[] Entero = new string[10]; //Numeros
public string[] Real = new string[11]; //Numero con punto flotante
public string[] Cadena = new string[1]; //Todo entre comillas
public string[] Tipo = new string[3];
public string[] OpSuma = new string[2];
public string[] OpMul = new string[2];
public string[] OpRelac = new string[4];
public string[] OpOr = new string[1];
public string[] OpAnd = new string[1];
public string[] OpNot = new string[1];
public string[] OpIgualdad = new string[2];
public string[] PuntoComa = new string[1];
public string[] Coma = new string[1];
public string[] ParentesisOpen = new string[1];
public string[] ParentesisClose = new string[1];
public string[] CorcheteOpen = new string[1];
public string[] CorcheteClose = new string[1];
public string[] Igual = new string[1];
public string[] _if = new string[1];
public string[] _while = new string[1];
public string[] _return = new string[1];
public string[] _else = new string[1];
public string[] Terminal = new string[1];
```

Tras esto se llena cada arreglo como se aclaro anteriormente con algunas modificaciones menores que normalmente veriamos normales en algunos lenguajes de programacion como la integracion de «_» a los identificadores que es algo bastante comun.

```

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", "_"};
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", "."};
Cadena = new string[1] {" "};
Tipo = new string[3] {"int", "float", "void"};
OpSuma = new string[2] {"+", "-"};
OpMul = new string[2] {"*", "/"};
OpRelac = new string[4] {"<", "<=", ">", ">="};
OpOr = new string[1] {"||"};
OpAnd = new string[1] {"&&"};
OpNot = new string[1] {"!"};
OpIgualdad = new string[2] {"==", "!="};
PuntoComa = new string[1] {";"};
Coma = new string[1] {","};
ParentesisOpen = new string[1] {"("};
ParentesisClose = new string[1] {")"};
CorcheteOpen = new string[1] {"{"};
CorcheteClose = new string[1] {"}"};
Igual = new string[1] {"="};
_if = new string[1] {"if"};
_while = new string[1] {"while"};
_return = new string[1] {"return"};
_else = new string[1] {"else"};
Terminal = new string[1] {"$"};

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"};

```

Como se observa hay un arreglo adicional el cual sera el encargado de mostrar los resultados correspondientes en cada uno de nuestros casos de validacion, esto se hace pensando a futuro pues, acostumbro hacerlo de esta manera para mas adelante, segun sea el caso, modificar los textos que muestra y optimizar el debuggin.

Ademas de lo anterior se crea un arreglo nuevo, que sera el responsable de separar las palabras para nosotros, esto con la finalidad de identificar comienzos y finales de palabras.

```

Separadores = new string[20] {"\r", "\n", "\t", " ", "+", "-", "*", "/", "<",
    ">", "(", ")", "{", "}", "=", "!", "|", "&", ",", ";"};

```

Tras las definicion de diccionarios comenzamos con programar la accion de un boton de ejecucion para procesar el texto del usuario y poder saber que ingreso, un codigo bastante simple y sin mucha complejidad, recorreremos el texto ingresado en la espera de un separados mientras guardamos lo que vamos leyendo, y en caso de encontrar separados significa que hay que procesar nuestra palabra y el separador por separado para poder regresar que tipos de datos son:

```

private void Compile_Click(object sender, EventArgs e)
{
    Results.Text = "";
    //Fase 1: Analisis Lexico
    string text = console.Text; //Todo el texto en cuestion
    string word = ""; //Esta es la palabra que vamos a procesar
    int i = 0;
    while (i < text.Length)//Mientras quede texto por recorrer
    {
        if (Separadores.Contains<string>(text[i].ToString()))
        {
            int x = -1;
            if(word != " " && word != "\t" && word != "")
            {
                x = AnalizadorLexico(word);
            }
            if (x > -1)
            {
                Lexico.Add(word);
                Results.Text += word + " -> " + Resultados[x] + "\r\n";
            }
            if(text[i] != ' ' && text[i] != '\t')
            {
                x = AnalizadorLexico(text[i].ToString());
            }
            else
            {
                x = -1;
            }

            if (x > -1)
            {
                Lexico.Add(text[i].ToString());
                Results.Text += text[i] + " -> " + Resultados[x] + "\r\n";
            }
            word = "";
        }
        else if(i == text.Length - 1)
        {
            word += text[i];
            int x = -1;
            if (word != " " && word != "\t" && word != "")
            {
                x = AnalizadorLexico(word);
            }
            if (x > -1)
            {
                Lexico.Add(word);
                Results.Text += word + " -> " + Resultados[x] + "\r\n";
            }
        }
        else
        {
            word += text[i];
        }
        i++;
    }
}

```

Se puede ver que el retorno de AnalizadorLexico es un numero, el cual es proporcional a

la posicion de un elemento en nuestro arreglo de Resultados.

Al entrar dentro del procesamiento Lexico se hace una comparacion de cada arreglo con la palabra en cuestion, para los Enteros, Reales e Identificadores la simple comparacion no es una manera fiable por lo que se usan los automatas.

```
if (Tipo.Contains<string>(word))
{
    return 4;
}
if (OpSuma.Contains<string>(word))
{
    return 5;
}
if (OpMul.Contains<string>(word))
{
    return 6;
}
if (OpRelac.Contains<string>(word))
{
    return 7;
}
if (OpOr.Contains<string>(word))
{
    return 8;
}
```

Los automatas:

```
//Reales y Numeros
int i = 0;
int opc = 0;
if(Real.Contains<string>(word[0].ToString()))
{
    if (word[0] == '.')
    {
        opc = 2;
    }
    else
    {
        opc = 1;
    }
    i++;
    while (i < word.Length)
    {
        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;
    default:
        break;
}
i++;
}

```

Se hace de esta manera para que al termino del procesamiento de la palabra completa tengamos 3 casos posibles, opc = 1, donde la palabra es un Entero, opc = 2, donde la palabra es un Real y por ultimo opc = -1, donde la palabra no es ninguno de los 2 y en general, no es nada aceptable, es decir, es un ERROR.

```

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

```

Ademas retornamos su valor correspondiente en el arreglo de Resultados e incluimos el error como un elemento mas.

Conclusiones.

Para finalizar la realizacion del Analizador Lexico realizamos la prueba de un par de codigos basicos para ver si el programa es capaz de disernir entre todos los elementos que yo escriba y ademas el programa sea capaz de identificar.

El codigo que se va a probar:

```

int main()
{
    while(numero ==1)
    {

    }

    flotante = 2.5;
}

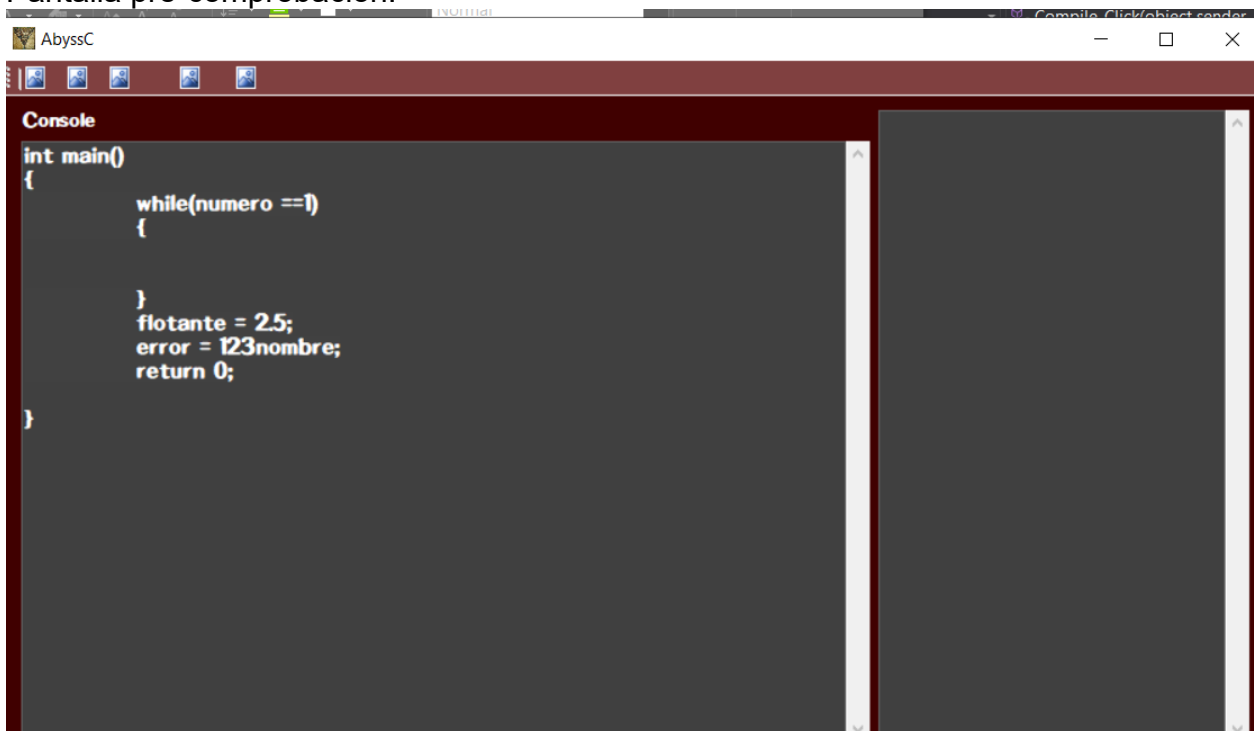
```



```
error = 123nombre;  
return 0;
```

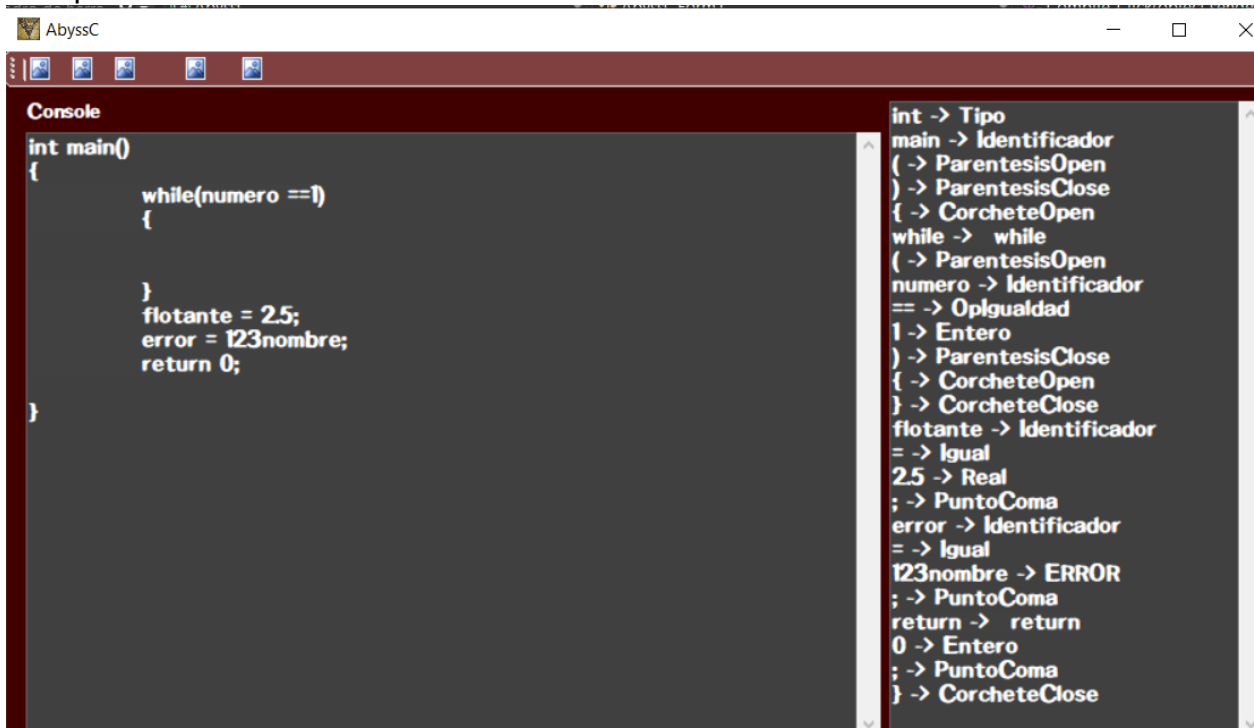
```
}
```

Pantalla pre-comprobacion:



```
int main()  
{  
    while(numero ==1)  
    {  
  
    }  
    flotante = 2.5;  
    error = 123nombre;  
    return 0;  
}
```

Comprobacion:



```
int main()  
{  
    while(numero ==1)  
    {  
  
    }  
    flotante = 2.5;  
    error = 123nombre;  
    return 0;  
}
```

```
int -> Tipo  
main -> Identificador  
( -> ParentesisOpen  
) -> ParentesisClose  
{ -> CorcheteOpen  
while -> while  
( -> ParentesisOpen  
numero -> Identificador  
== -> Oplgualdad  
1 -> Entero  
) -> ParentesisClose  
{ -> CorcheteOpen  
} -> CorcheteClose  
flotante -> Identificador  
= -> Igual  
2.5 -> Real  
; -> PuntoComa  
error -> Identificador  
= -> Igual  
123nombre -> ERROR  
; -> PuntoComa  
return -> return  
0 -> Entero  
; -> PuntoComa  
} -> CorcheteClose
```

Podemos apreciar que efectivamente funciona y es capaz de diferenciar incluso los Igual asingativos de los igual comparativos, ademas de identificar un error y diferenciar los tipos de numero Real de Entero asi como los respectivos identificadores.