UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Inteligencia Artificial

Reporte de práctica

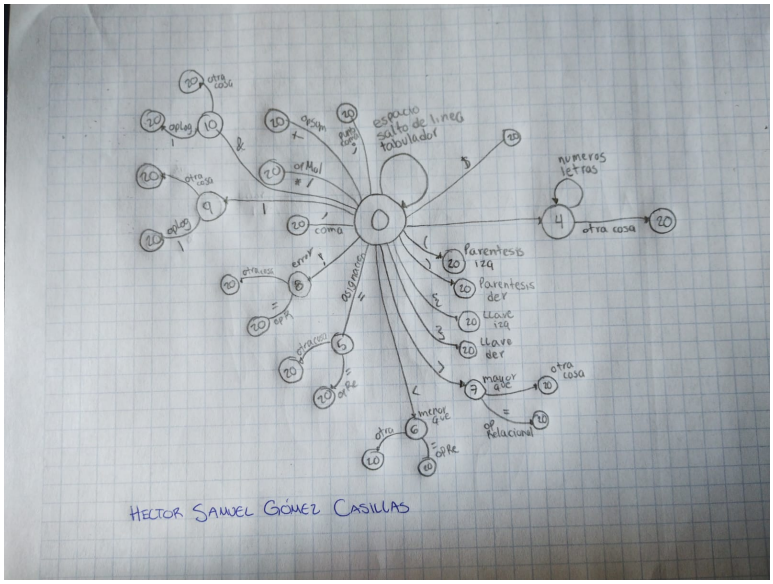
Nombre del alumno:	Héctor Samuel Gómez Casillas
Profesor:	Erasmo Gabriel Martínez Soltero
Título de la práctica:	"Tarea 2. Analizador Lexico"
Fecha:	12 septiembre 2023

Introducción

La actividad presenta una implementacion más desarrollada del programa proporcionado por el profesor, en el que se programó un automata para analizar una cadena de entrada y clasificar sus tokens conforme a las reglas establecidas.

Metodología

La metodología empleada para resolver esta problemática se basa en el uso de condicionales 'if' en el lenguaje de programación C#. A continuación, se describe el método para validar las cadenas de texto conforme lo haria el siguiente automata:



Estado 0: Este es el estado inicial del automata,y en el cual se tomará los diferentes caminos dependiendo de la entrada y las condiciones establecidas en las reglas.

Primeramente se valida las entradas de espacios en blanco (se queda en el estado 0), entradas digitales (cambia a estado 3), entradas solamente con letras y/o '_' (cambia a estado 4) y por ultimo si es el final de cadena, caracterizado por el digito '$' (pasa a estado de aceptacion y se actualiza su token y numero).

```
if (estado == 0)
{
    if (char.IsWhiteSpace(cadena[indice]))
    {
        estado = 0;
    }
    else if (char.IsDigit(cadena[indice]))
    {
        estado = 3;
        lexema += cadena[indice];
    }
    else if (char.IsLetter(cadena[indice]) || cadena[indice] == '_')
    {
        estado = 4;
        lexema += cadena[indice];
    }
    else if (cadena[indice] == '$')
    {
        estado = 20;
        lexema += cadena[indice];
        token = "pesos";
        numero = 18;
```

Posteriormente se analiza si la entrada es '=' para despues validar si es un token de Asignacion o algo más (pasa a estado 8 y se actualiza su token y numero) y se valida si la entrada son '(',')','{' ó '}'(cambia a estado 4,5,6 y 7 respectivamente y se actualiza su token junto con su numero).

```
else if (cadena[indice] == '=')
{
    lexema += cadena[indice];
    token = "asignación";
    numero = 8;
    estado = 5;
}
else if (cadena[indice] == '(')
{
    token += "parentesisIz";
    numero = 4;
    estado = 20;
    lexema += cadena[indice];
}
else if (cadena[indice] == ')')
{
    token += "paretensisDer";
    numero = 5;
    estado = 20;
    lexema += cadena[indice];
}
else if (cadena[indice] == '{')
{
    token += "llaveIz";
    numero = 6;
    estado = 20;
    lexema += cadena[indice];
}
else if (cadena[indice] == '}')
{
    token += "llaveDer";
    numero = 7;
    estado = 20;
    lexema += cadena[indice];
}
```

A continuacion se sigue validando si la entrada es '>' o '<' para despues verificar si se trata de un token de Operacion Relacional o algo más (se pasa a estado 6 y 7 respectivamente y se actualiza su token junto con su numero), tambien se valida si la entrada es una ',' o ';' (pasa a estado de aceptacion y se actualiza su token y numero respectivamente).

```
else if (cadena[indice] == '>')
{
    token += "opRelacional";
    numero = 17;
    estado = 6;
    lexema += cadena[indice];
}
else if (cadena[indice] == '<')
{
    token += "opRelacional";
    numero = 17;
    estado = 7;
    lexema += cadena[indice];
}
else if (cadena[indice] == ',')
{
    token += "coma";
    numero = 3;
    estado = 20;
    lexema += cadena[indice];
}
else if (cadena[indice] == ';')
{
    token += "puntoComa";
    numero = 2;
    estado = 20;
    lexema += cadena[indice];
}
```

Ya por ultimo en el estado 0 se verifica si la entrada es '+' o '-' para validar si es un token de Operacion Suma (pasa a estado de aceptacion y se actualiza su token y numero), se valida si la entrada es '*' o '/' para validar si es un token de Operacion Multiplicacion (pasa a estado de aceptacion y se actualiza su token y numero), se valida si la entrada es '—' (se pasa al estado 9 y se actualiza su numero), posteriormente se valida se la entrada es '&'(se pasa al estado 10 y se actualiza su numero), finalmente si la entrada no coincide con ninguno de los posibles caminos para el estado 0 se clasifica como un token 'error'.

```
else if (cadena[indice] == '+' || cadena[indice] == '-')
{
    token += "opSuma";
    numero = 14;
    estado = 20;
    lexema += cadena[indice];
}
else if (cadena[indice] == '*' || cadena[indice] == '/')
{
    token += "opMul";
    numero = 16;
    estado = 20;
    lexema += cadena[indice];
}
else if (cadena[indice] == '!')
{
    numero = 17;
    estado = 8;
    lexema += cadena[indice];
}
else if (cadena[indice] == '|')
{
    numero = 17;
    estado = 9;
    lexema += cadena[indice];
}
else if (cadena[indice] == '&')
{
    numero = 17;
    estado = 10;
    lexema += cadena[indice];
}
else
{
    estado = 20;
    token = "error";
    lexema = cadena[indice].ToString();
}
indice++;
```

A continuacion se presenta el estado 3: En el que si la entrada es un digito, se clasifica como 'constante' y se actualiza su numero, en caso contrario se pasa al estado de aceptacion.

```
else if (estado == 3)
{
    if (char.IsDigit(cadena[indice]))
    {
        estado = 3;
        token = "constante";
        numero = 13;
        indice++;
        lexema += cadena[indice];
    }
    else
    {
        estado = 20;
        // ...
    }
}
```

A continuacion se presenta el estado 4: En el que si la entrada es un digito, una letra o un '_' se clasifica como 'id y se actualiza su numero, en caso contrario se pasa al estado de aceptacion.

```
else if (estado == 4)
{
    if (char.IsDigit(cadena[indice]) || char.IsLetter(cadena[indice]) || cadena[indice] == '_')
    {
        estado = 4;
        lexema += cadena[indice];
        token = "id";
        numero = 1;
        indice++;
    }
    else
    {
        estado = 20;
    }
}
```

A continuacion se presenta el estado 5: En el que si la entrada es diferente de '=' se pasa al estado de aceptacion, en caso contrario se clasifica como 'opRelacional' y se actualiza su numero.

```
else if (estado == 5)
{
    if (cadena[indice] != '=')
    {
        estado = 20;
    }
    else
    {
        estado = 20;
        lexema += cadena[indice];
        token = "opRelacional";
        numero = 17;
        indice++;
    }
}
```

A continuacion se presenta el estado 6: En el que si la entrada es diferente de '=' se pasa al estado de aceptacion, en caso contrario se clasifica como 'opRelacional' y se actualiza su numero.

```
else if (estado == 6)
{
    if (cadena[indice] != '=')
    {
        estado = 20;
    }
    else
    {
        estado = 20;
        lexema += cadena[indice];
        token += "opRelacional";
        indice++;
    }
}
```

A continuacion se presenta el estado 7: En el que si la entrada es diferente de '=' se pasa al estado de aceptacion, en caso contrario se clasifica como 'opRelacional' y se actualiza su numero.

```
else if (estado == 7)
{
    if (cadena[indice] != '=')
    {
        estado = 20;
    }
    else
    {
        estado = 20;
        lexema += cadena[indice];
        token += "opRelacional";
        indice++;
    }
}
```

A continuacion se presenta el estado 8: En el que si la entrada es diferente de '=' se pasa al estado de aceptacion, en caso contrario se clasifica como 'opRelacional' y se actualiza su numero.

```
else if (estado == 8)
{
    if (cadena[indice] != '=')
    {
        estado = 20;
    }
    else
    {
        estado = 20;
        lexema += cadena[indice];
        token += "opRelacional";
        indice++;
    }
}
```

A continuacion se presenta el estado 9: En el que si la entrada es diferente de '—' se pasa al estado de aceptacion, en caso contrario se clasifica como 'opLogica' y se actualiza su numero.

```
else if (estado == 9)
{
    if (cadena[indice] != '|')
    {
        estado = 20;
    }
    else
    {
        estado = 20;
        lexema += cadena[indice];
        token += "opLogica";
        indice++;
    }
}
```

Por ultimo se presenta el estado 10: En el que si la entrada es diferente de '&' se pasa al estado de aceptacion, en caso contrario se clasifica como 'opLogica' y se actualiza su numero.

```
else if (estado == 10)
{
    if (cadena[indice] != '&')
    {
        estado = 20;
    }
    else
    {
        estado = 20;
        lexema += cadena[indice];
        token += "opLogica";
        indice++;
    }
}
```

Despues de agregan todos los tokens a la lista de 'elementos' con sus datos correspondientes.

```
    elementos.Add(new Dictionary<string, object>
{
    { "token", token },
    { "lexema", lexema },
    { "numero", numero }
});
}
```

En la lista de elementos se buscan casos especificos de palabras reservadas para cambiar su atributos.

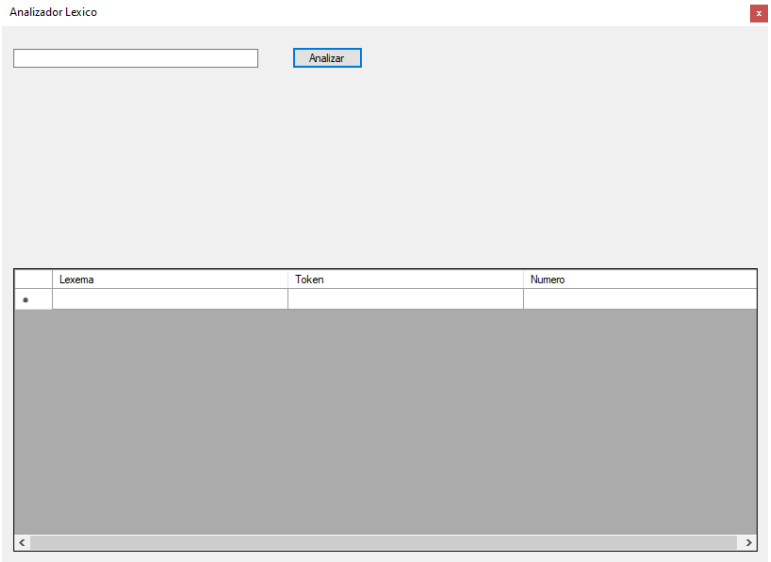
```
foreach (var elemento in elementos)
{
    if (elemento["lexema"].ToString() == "if")
    {
        elemento["token"] = "condicional SI";
        elemento["numero"] = 9;
    }
    if (elemento["lexema"].ToString() == "while")
    {
        elemento["token"] = "while";
        elemento["numero"] = 10;
    }
    if (elemento["lexema"].ToString() == "return")
    {
        elemento["token"] = "return";
        elemento["numero"] = 11;
    }
    if (elemento["lexema"].ToString() == "else")
    {
        elemento["token"] = "else";
        elemento["numero"] = 12;
    }

    //Console.WriteLine(elemento);
}
```

Por ultimo se actualiza el dataGrid para ver el resultado del analizador lexico.

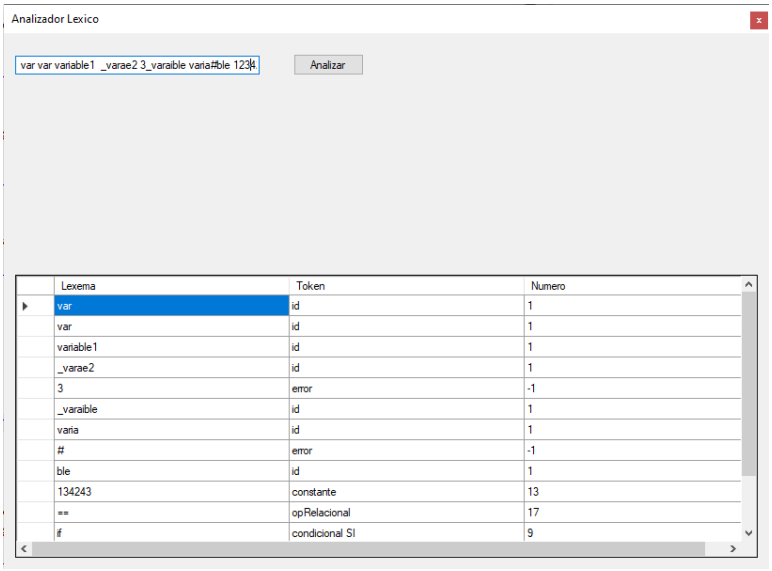
```
// Limpia el DataGridView antes de mostrar los resultados
dataGridView1.Rows.Clear();

// Agrega los resultados al DataGridView
foreach (var elemento in elementos)
{
    string lexema = elemento["lexema"].ToString();
    string token = elemento["token"].ToString();
    int numero = (int)elemento["numero"];

    dataGridView1.Rows.Add(lexema, token, numero);
}
```

Resultados

Ventana Principal



Resultado



Conclusiones

La implementacion de un programa que simule un Automata es muy interesante, este analizador lexico es simple y sencillo ya que no valida entradas muy complejas, por lo tanto la implementacion de muchos 'if' dentro de un ciclo 'while' es algo optimo viendo el alcance del programa.

Referencias

- Martinez Soltero E. (2021). ."EjemploLexico"(1.0) Python.