



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 Sintactico Completo

Fecha de Entrega:

31 de Enero del 2022

Introduccion.

Para realizar el analisis sintactico y mas adelante poder realizar un analisis semantico, se deben realizar un par de clases nuevas, asi como hacer clases que heredan del objeto que se almacenara en nuestra pila.

Es decir, para esta práctica sera necesario crear una clase ElementoPila (EP) y modificar la clase pila para que acepte objetos de este tipo en lugar de enteros.

Necesitaras crear 3 clases más, las cuales heredan de ElementoPila, las clases son:

- Terminal -> (que llamaremos T)
- No terminal -> (que llamaremos NT)
- Estado -> (que llamaremos E)

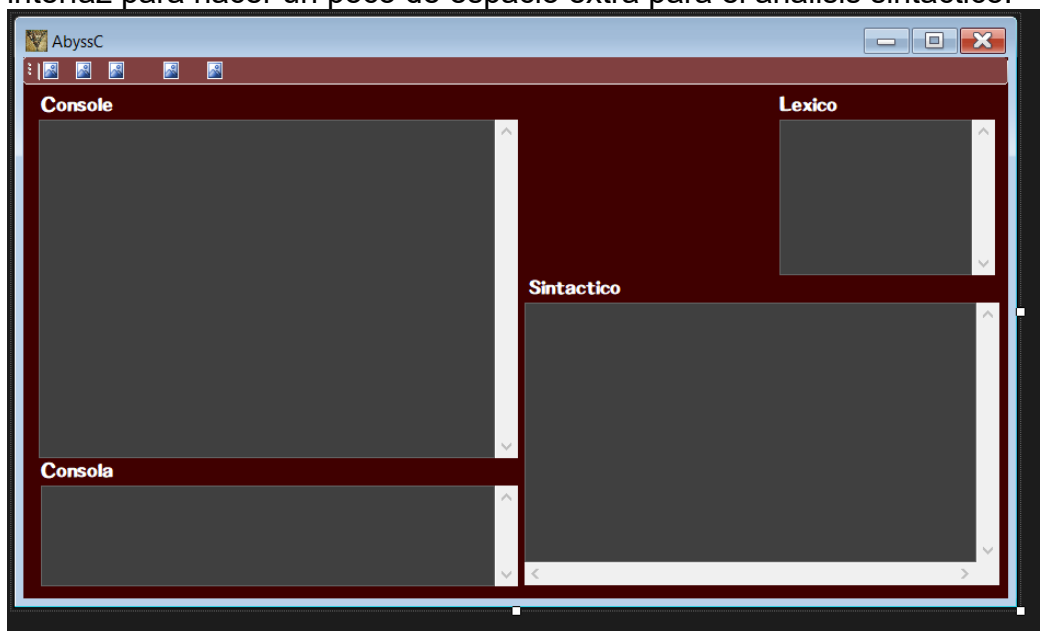
Entonces para esta actividad sera necesario hacer uso de una tabla de un analizador sintactico LR:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	identif	entero	real	cadena	tipo	opSuma	opMul	opRelac	opOr	opAnd	opNot	opIgualda;	,	()	{	}	=	#	while	return	else	\$	programa	Definicion	DefVar	Listalar	DefFunc	Parametro	ListParam		
1					d5																		r2	1	2	3	4		6			
2																							r0									
3					d5																		r1									
4					r4																		r2			7	3	4		6		
5	d8																						r4									
6					r5																		r5									
7																							r3									
8																														9		
9													r7	d10	d11																	
10	d13												d12																			
11					d15																										14	
12	r6				r6																											
13																																
14													r7	d10			r10		r6		r6	r6		r6						16		
15	d18																															
16																																
17													r8																			
18																																
19														d22			r12		d20													
20	d27				r9																											
21					d5																											
22					d32																											
23																																
24	d27				d5																											
25	r17				r17																											
26	r18				r18																											
27																																
28																																
29																																
30	d46	d47	d48	d49		d42																										
31																																
32	d51																															
33					r14																											
34																																
35	d46	d47	d48	d49		d42																										
36	d46	d47	d48	d49		d42																										
37	d46	d47	d48	d49		d42																										

*Nota: considerar que lo mostrado aqui solo es una parte de la tabla

Ademas de esto, se debera crear un automata para que se haga el analisis sintactico de manera similar a como se realizaria a mano.

Para comenzar a trabajar e implementar los nuevos metodos y algoritmos se modifica el interfaz para hacer un poco de espacio extra para el analisis sintactico.



Desarrollo.

En seguida, se crea un arreglo bidimensional para introducir nuestra tabla LR con la que realizaremos el analisis.

[illegible]

Haciendo uso de este arreglo bidimensional vamos a saber que hacer tras cada paso, ya sea un desplazamiento (numeros positivos), o una reduccion (numeros negativos), para cada reduccion se programa dentro de un automata de la siguiente manera:

```

if (accion < 0)
{
    //Results_Sintactico.Text += accion.ToString() + "\r\n";
    switch (accion)
    {
        case -1:
            Results_Console.Text += "Análisis sintactico completado con satisfaccion";
            parsing = false;
            i--;
            break;
        case -2:
            //R1
            PopPila(1);
            fila = Int32.Parse(pila.Top().cadena);
            elemento = new EP("<programa>");

            pila.Push(elemento);
            //Results_Sintactico.Text += elemento.cadena;

            accion = LR[fila, 24];
            //Results_Sintactico.Text += accion.ToString();
            elemento = new EP(accion.ToString());

            pila.Push(elemento);
            //Results_Sintactico.Text += elemento.cadena;

            i--;
            break;
        case -3:
            //R2

```

Aquí podemos ver que -1 manda al estado de éxito, en cambio, a partir del -2 el resto de reducciones aplican su regla correspondiente, todas las reducciones hacen uso de la fun-

ción «PopPila(numero);» donde el numero es la cantidad de elementos que debemos de restar, de manera que quitamos la cantidad de elementos que dicta la ecuacion y colocamos en su lugar el elemento que la misma regla estipula.

```
public void PopPila(int tokens)
{
    int PopTokens, i = 0;
    PopTokens = tokens * 2;
    while(i < PopTokens)
    {
        pila.Pop();
        i++;
    }
}
```

*Como se menciona, el popeo de elementos dentro de nuestra pila se hace con el uso de una funcion que popea «tokens» cantidad de elementos.

Esto se hace de esta manera para poder conseguir en seguida las acciones a realizar, las acciones son las coordenadas en que nos vamos a desplazar dentro de nuestro arreglo bidimensional con las instrucciones LR, y tras hacer todo esto, almacenamos con un «push» dentro de nuestra pila, esto ira almacenando todo nuestro analisis de manera que podemos mostrarlo por cada ciclo que haga para ver como aumenta o se reduce.

```
$0int5main8
$0int5main8(I1
$0int5main8(I1<Parametros>14
$0int5main8(I1<Parametros>14)I7
$0int5main8(I1<Parametros>14)I7(20
```

De una manera similar a esta, donde podemos observar lo que entra y sale de la pila por cada accion realizada.

En seguida hacemos unos retoques a nuestra pila que anteriormente no soportaba objetos de clase EP, esto con el fin de poder almacenar EP y sus derivados.

```
namespace AbyssC
{
    public class Pila
    {
        //Variables
        public List<EP> pila = new List<EP>();

        //Metodos

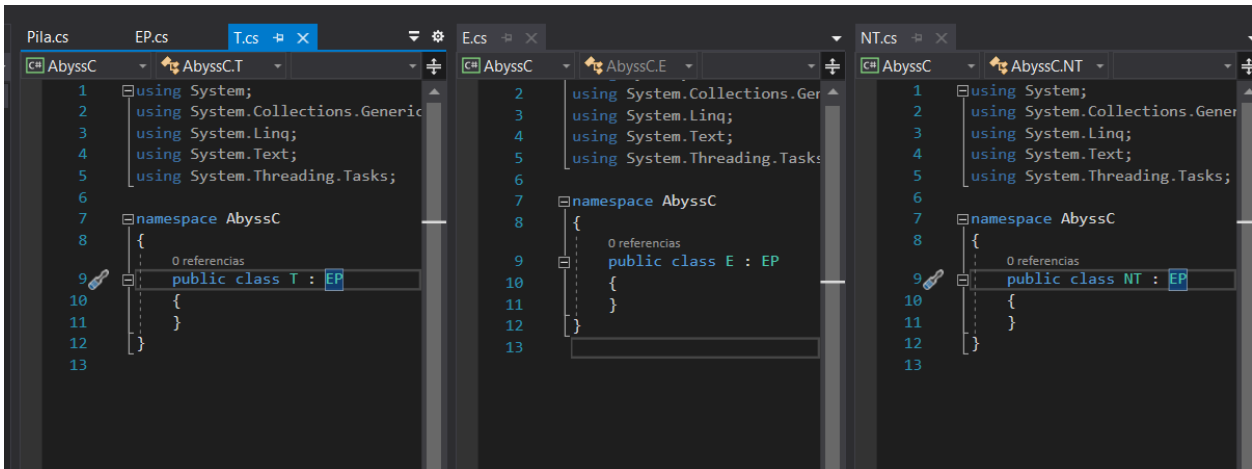
        public virtual void Push(EP top)
        {
            pila.Add(top);
        }

        public virtual void Pop()
        {
            pila.RemoveAt(pila.Count - 1);
        }

        public virtual EP Top()
        {
            return pila.Last<EP>();
        }

        public virtual string Show()
        {
            string cadena = "";
            foreach(EP element in pila)
            {
                cadena += element.cadena;
            }
            return cadena;
        }
    }
}
```

Tras esto, se crean las clases que heredan de EP, estas clases serán T (terminales), E (estado) y NT (no terminales), los objetos de cada tipo se crearán respecto a las acciones resultantes de cada uno de los elementos pila que analicemos más adelante.



Para finalizar agregaré una función que me deje saber detrás de qué elemento sucede el error, de momento esto no es más que un agregado que se tratará de mejorar más adelante.

```
public void Errores(int error)
{
    Results_Console.Text += "Error sintactico, analisis fallido: " + error.ToString() + "\r\n";
    string msgError = "";
    if(error < 4)
    {
        msgError = ";";
    }
    else if(error < 12)
    {
        msgError = "Variable";
    }
    else if (error == 23)
    {
        msgError = "}";
    }
    Results_Console.Text += "Syntax Error ' " + MensajesError[error] + " ' " + "\r\n";
    Results_Console.Text += "Se esperaba -> ' " + msgError + " ' ";
}
```

Una vez explicados los pasos, simplemente introducimos nuestro programa en el analizador sintáctico y comenzamos a discriminar el interior de este, de manera que comenzará a aplicar las reglas y desplazamientos que indique nuestro arreglo LR. En seguida, gregará el proceso a nuestra pila y para concluir, en caso de detectar alguna inconsistencia dentro de nuestro código que no sea admisible por nuestro compilador sintáctico, procederá a saltar el error y el punto del código en el que detectó el error.

```
public void AnalizadorSintactico(string programa)
{
    int fila, columna;
    pila.Push(new EP("$"));
    pila.Push(new EP("0"));

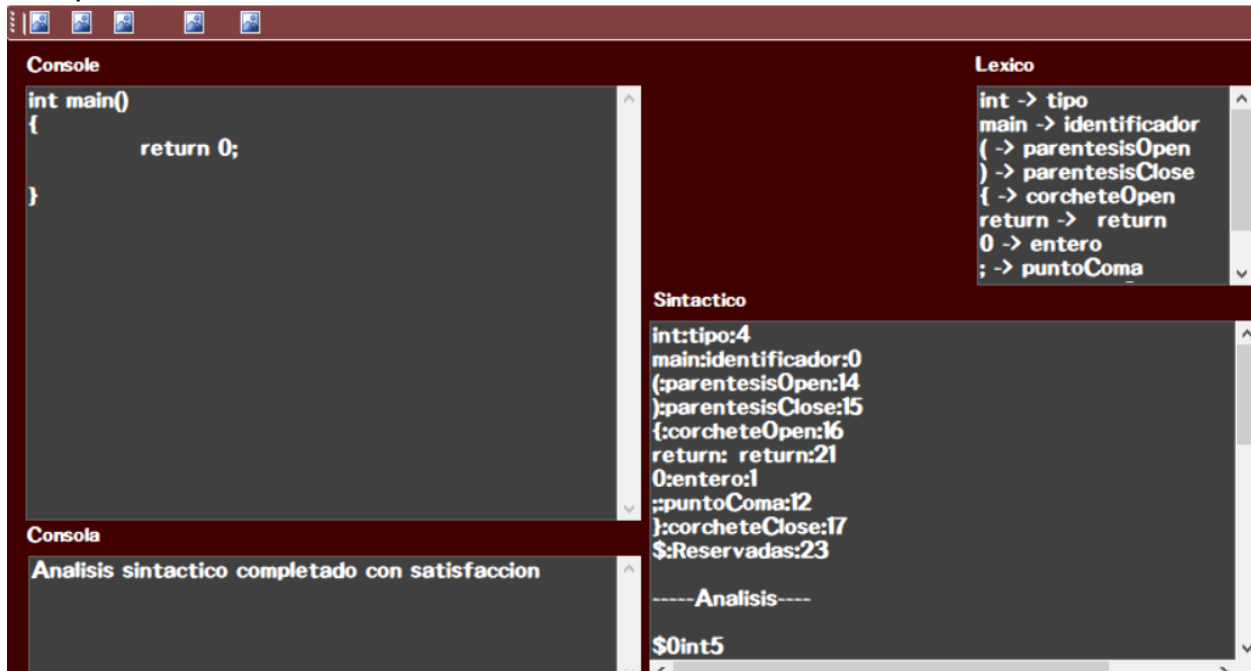
    //El numero que está en el tope de la fila + el número del identificador
    //int|Reservadas|4 main|Identificador|0 (|Simbolo|14 )|Simbolo|15 {|Simbolo|16 }|Simbolo|17
    EP elemento;
    Results_Sintactico.Text += "\r\n" + "-----Analisis-----" + "\r\n\r\n";
    int i = 0, j, control, accion = 0;
    char c;
    string data = "", tipo = "0", token = "";
    bool llave = false, parcing = true;
    while (parcing)
    {

```

Conclusiones.

Para finalizar, comprobamos las entradas y salidas de nuestro analizador sintactico, asi como el estado de la pila durante todo el proceso de analisis.

Comprobacion:



The screenshot shows a compiler interface with three main panels:

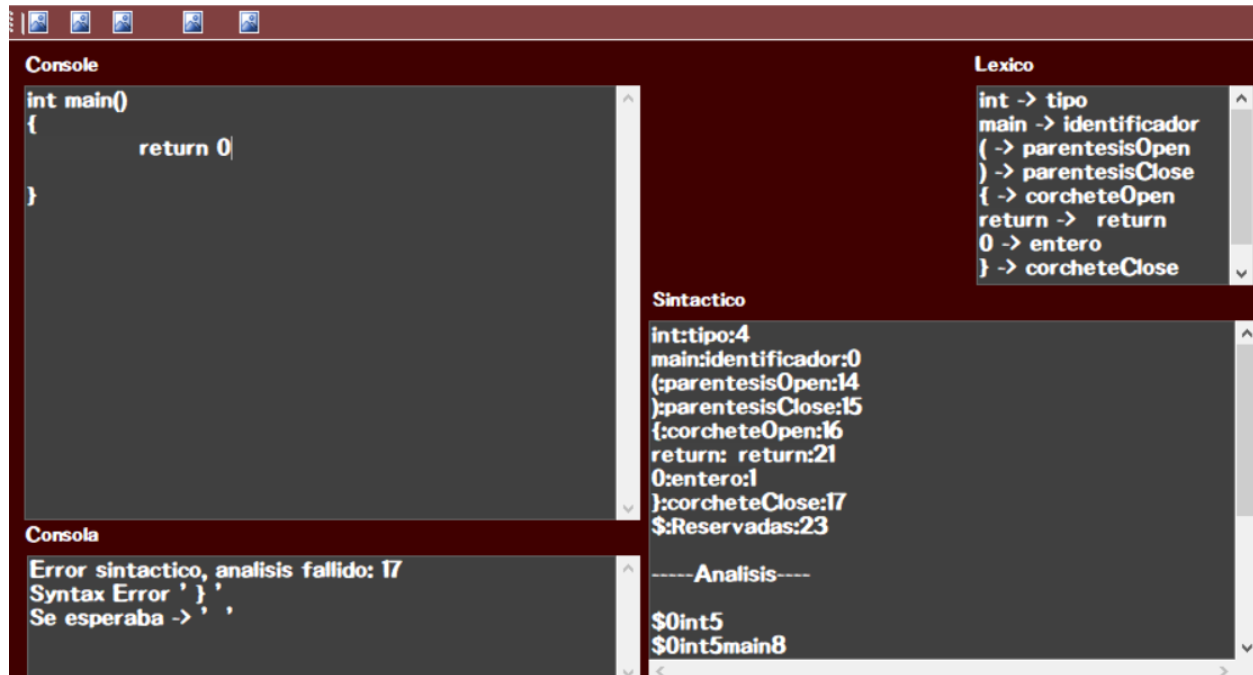
- Console:** Displays the source code: `int main() { return 0; }`
- Lexico:** Shows the lexical analysis results, mapping tokens to their categories: `int -> tipo`, `main -> identificador`, `(-> parentesisOpen`, `) -> parentesisClose`, `{ -> corcheteOpen`, `return -> return`, `0 -> entero`, and `;-> puntoComa`.
- Sintactico:** Shows the syntactic analysis results, including token positions and the stack state: `int:tipo:4`, `main:identificador:0`, `(:parentesisOpen:14`, `):parentesisClose:15`, `{:corcheteOpen:16`, `return: return:21`, `0:entero:1`, `;;puntoComa:12`, `};corcheteClose:17`, and `$.Reservadas:23`. Below this, it shows the stack state: `----- Analisis -----` and `$0int5`.

Contenido en la pila:

```
$0int5
$0int5main8
$0int5main8(11
$0int5main8(11<Parametros>14
$0int5main8(11<Parametros>14)17
$0int5main8(11<Parametros>14)17{20
$0int5main8(11<Parametros>14)17{20return30
$0int5main8(11<Parametros>14)17{20return30047
$0int5main8(11<Parametros>14)17{20return30<Termino>44
$0int5main8(11<Parametros>14)17{20return30<Expresion>40
$0int5main8(11<Parametros>14)17{20return30<ValorRegresa>39
$0int5main8(11<Parametros>14)17{20return30<ValorRegresa>39;57
$0int5main8(11<Parametros>14)17{20<Sentencia>26
$0int5main8(11<Parametros>14)17{20<DefLocal>24
$0int5main8(11<Parametros>14)17{20<DefLocal>24<DefLocales>34
$0int5main8(11<Parametros>14)17{20<DefLocales>23
$0int5main8(11<Parametros>14)17{20<DefLocales>23}33
$0int5main8(11<Parametros>14)17<BloqFunc>19
$0<DefFunc>6
$0<Definicion>3
$0<Definicion>3<Definiciones>7
$0<Definiciones>2
$0<programa>1
$0<programa>1
```

***Observamos un analisis exitoso, sin errores ni problemas, lo que deberia ser siem-**

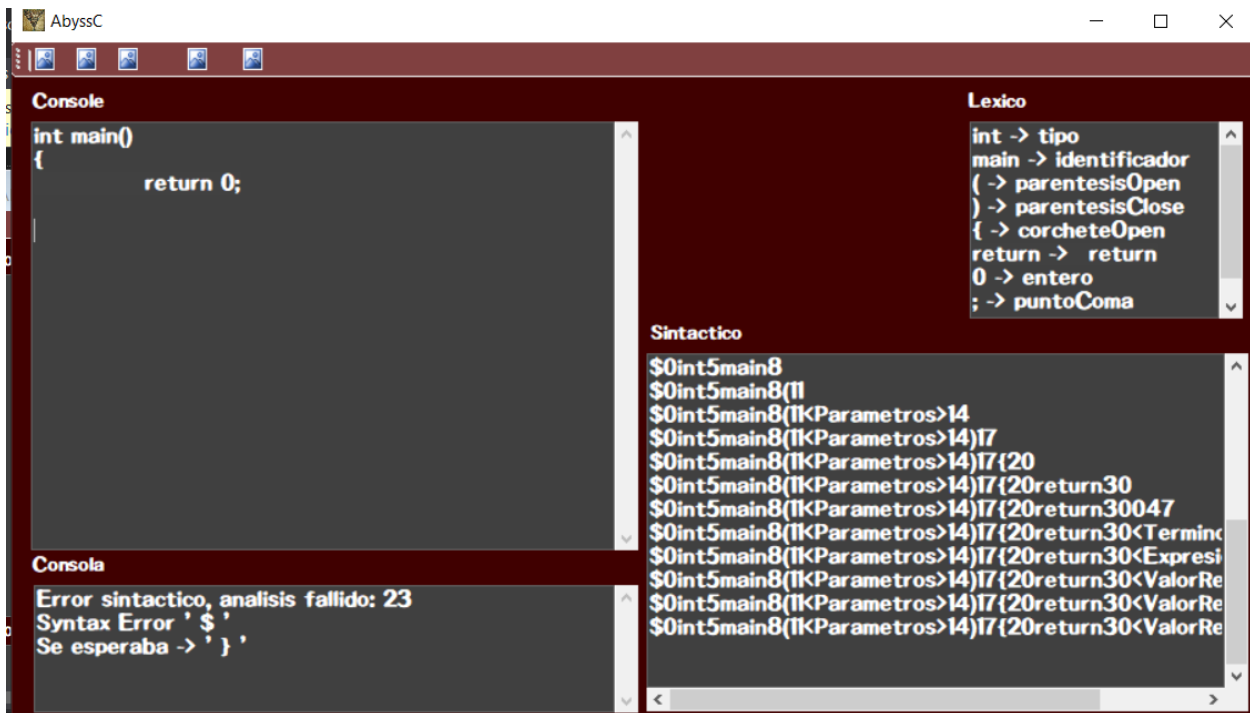
pre que una entrada no contenga errores como los siguientes.



Contenido en la pila:

```
$0int5
$0int5main8
$0int5main8(11
$0int5main8(11<Parametros>14
$0int5main8(11<Parametros>14)17
$0int5main8(11<Parametros>14)17{20
$0int5main8(11<Parametros>14)17{20return30
$0int5main8(11<Parametros>14)17{20return30047
$0int5main8(11<Parametros>14)17{20return30047
```

***El error en código fue evidente y se puede comprobar con el estado de la pila que no pudo reducir y dejó la pila en ese estado de error.**



Contenido en la pila:

```

$0int5
$0int5main8
$0int5main8(11
$0int5main8(11<Parametros>14
$0int5main8(11<Parametros>14)17
$0int5main8(11<Parametros>14)17{20
$0int5main8(11<Parametros>14)17{20return30
$0int5main8(11<Parametros>14)17{20return30047
$0int5main8(11<Parametros>14)17{20return30<Termino>44
$0int5main8(11<Parametros>14)17{20return30<Expresion>40
$0int5main8(11<Parametros>14)17{20return30<ValorRegresa>39
$0int5main8(11<Parametros>14)17{20return30<ValorRegresa>39;57
$0int5main8(11<Parametros>14)17{20return30<ValorRegresa>39;57

```

***Aqui la pila no termino de vaciarse, por lo que se puede deducir que efectivamente el error en codigo fue detectado y retornado.**