



UNIVERSIDAD AUTONOMA DE SAN LUIS POTOSI
FACULTAD DE INGENIERIA
AREA DE CIENCIAS DE LA COMPUTACIÓN
MATERIA LABORATORIO FUNDAMENTOS DE SOFTWARE DE
SISTEMAS



“Practica 02: Analizador léxico y sintáctico para lenguaje ensamblador de SIC XE”



Alumna

Sanjuanero Herrera Nohemí

Grupo:240902

Fecha:01/03/2022

Semestre: 2021-2022/II

Nombre del Profesor: Agustín Hernández García

1. Descripción de los componentes léxicos

```
CODOPF1
: 'FIX '
| 'FLOAT ' | 'HIO ' | 'NORM ' | 'SIO ' | 'TIO '
;
```

CODOPF1: Define los códigos de operación que pertenecen al formato uno.

```
CODOPF2
: 'ADDR ' | 'CLEAR ' | 'COMPR ' | 'DIVR ' | 'RMO '
| 'SHIFTL ' | 'SHIFTR ' | 'SUBR '
| 'SVC ' | 'TIXR '
;
```

CODOPF2: Define los códigos de operación que pertenecen al formato dos.

```
CODOPF3
: 'ADD ' | 'ADDF ' | 'AND ' | 'COMP '
| 'COMPF ' | 'DIV ' | 'DIVF ' | 'J ' | 'JEQ ' | 'JGT ' | 'JLT ' | 'JSUB '
| 'LDA ' | 'LDB ' | 'LDCH ' | 'LDF ' | 'LDL ' | 'LDS ' | 'LDT '
| 'LDX ' | 'LPS ' | 'MUL ' | 'MULF ' | 'MULR ' | 'OR '
| 'RD ' | 'RSUB '
| 'SSK ' | 'STA ' | 'STB ' | 'STCH ' | 'STF ' | 'STI ' | 'STL '
| 'STS ' | 'STSW ' | 'STT ' | 'STX ' | 'SUB ' | 'SUBF ' | 'TD ' | 'TIX ' | 'WD '
;
```

CODOPF3: Define los códigos de operación que pertenecen al formato tres.

```
REG
: 'A' | 'X' | 'L' | 'B' | 'S' | 'T'
| 'F' | 'CP' | 'SW'
;
```

REG: Registros validos en la SICXE

```
START
: 'START '
;
```

START: Identifica la etiqueta START (inicio de programa)

```
DIRECTIVA
: 'BYTE ' | 'WORD ' | 'RESB ' | 'RESW ' | 'BASE '
;
```

DIRECTIVA: Directivas validas en la SICXE

```
END
: 'END '
;
```

END: Identifica la etiqueta END (Fin de programa)

```
SEP
: ','
;
```

SEP: Separador entre ID y registro

```
F4
: '+'
;
```

F4: Símbolo de suma '+' identifica al formato cuatro

```
EBSILON
: ' '
;
```

EBSILON: Identifica el espacio

```
IND
: '@'
;
```

IND: Modo de direccionamiento indirecto

```
INM
: '#'
;
```

INM: Modo de direccionamiento Inmediato

```
CONSTHEX
: 'X' '(' ('0'..'9' | 'A'..'Z')+' ')
```

CONSTHEX: Constante en hexadecimal para directiva BYTE

```
CONSTCAD
: 'C' '(' ('a'..'z' | 'A'..'Z')+' ')
```

CONSTCAD: Constante en hexadecimal para directiva BYTE

```
NUMERO2
: ('0'..'9')+ (('0'..'9' | 'A'..'F')+'H')
```

NUMERO2: para identificar números y números hexadecimales

```

ID
: ('A'..'Z' | '0'..'9')+
| ('A'..'Z' | '0'..'9')+
;

```

ID: Para identificadores (Etiquetas)

```

FINL
: '\n'
;

```

FINL: Identificar el fin de la instrucción

```

WS
: (' ' | '\n' | '\t')+ {Skip();};

```

WS: para identificar secuencias de escape

2. Explicación de las reglas gramaticales utilizadas.

Las reglas gramaticales se desarrollaron en base a la gramática de la SICXE.

Programa

```
:programa
  : inicio proposiciones fin
  ;
```

Inicio: Existen dos posibles caminos, en el primer caso la gramática aceptaría una instrucción como la siguiente:

NUEVO START 0

```
:inicio
  :etiqueta START NUMERO2 FINL
  | proposicion
  ;
```

Fin: Existen dos posibles caminos. La entrada puede o no existir después de END

```
:fin
  :END entrada FINL
  |END entrada
  ;
```

Entrada: puede o no existir después de END esto en regla fin. Esto se logra agregando el símbolo cero o una instancia (?)

```
:entrada
  :(ID)?
  ;
```

Proposiciones: Permite tener varias proposiciones. Esta instrucción originalmente tiene recursividad por la izquierda la cual fue eliminada con la herramienta ANTLRWORKS

```
:proposiciones
  : (proposicion)( (proposicion)*)
  ;
```

Proposición: Dos posibles caminos, Instrucción o directiva.

```
:proposicion
  :instruccion
  |directiva
  ;
```

Instrucción: puede o no existir la etiqueta en una instrucción, pero siempre debe tener código de operación y fin de instrucción.

```
:instruccion
  :etiqueta opinstruccion FINL
  ;
```

Directiva: puede o no existir la etiqueta en la directiva, pero debe tener el tipo de directiva (BYTE ' | 'WORD ' | 'RESB ' | 'RESW ' | 'BASE ') el operador de la directiva y el fin de línea.

```
directiva
:etiqueta tipoDirectiva opDirectiva FINL
;
```

tipoDirectiva: incluye a (BYTE ' | 'WORD ' | 'RESB ' | 'RESW ' | 'BASE ')

```
tipoDirectiva
:DIRECTIVA
;
```

Etiqueta: puede o no existir la etiqueta en una instrucción o en una directiva

```
:etiqueta
:(ID)?
;
```

Opinstruccion: hace uso de la regla formato

```
:opinstruccion
:formato?|
;
```

Formato: incluye los formatos disponibles que van desde el uno hasta el cuatro.

```
Formato
:formatoUno
|formatoDos
|formatoTres
|formatoCuatro
;
```

formatoUno: incluye el formato uno (solo código de operación)

```
formatoUno
:CODOPF1
;
```

formatoDos: incluye el formato dos (cuatro posibles opciones)

```
:formatoDos
:CODOPF2 NUMERO2
|CODOPF2 REG
|CODOPF2 REG SEP REG
|CODOPF2 REG SEP NUMERO2
;
```

formatoTres: Incluye simple, indirecto e inmediato.

```
:formatoTres
:simple3|
|indirecto3
|inmediato3
;
```

Formato4: Incluye las mismas opciones que el formato tres solo que debe tener el signo '+' antes del CODOP, ya que esto identifica el formato cuatro.

```
formatoCuatro
: F4 (simple3
| indirecto3
| inmediato3)
;
```

Simple3: incluye el formato simple (cuatro posibles opciones)

```
simple3
: CODOPF3 ID
| CODOPF3 NUMERO2
| CODOPF3 NUMERO2 SEP REG
| CODOPF3 ID SEP REG
;
```

Indirecto3: incluye el formato indirecto (dos posibles opciones)

```
indirecto3
: CODOPF3 IND NUMERO2
| CODOPF3 IND ID
;
```

Inmediato3: incluye el formato inmediato (dos posibles opciones)

```
inmediato3
: CODOPF3 INM NUMERO2
| CODOPF3 INM ID
;
```

opDirectiva: puede ser un numero (decimal / hexadecimal), constante hexadecimal, constante cadena o un identificador.

```
opDirectiva
: NUMERO2
| CONSTHEX
| CONSTCAD
| ID
;
```

Registro: incluye REG (contiene los registros validos para la SICXE)

```
registro
: REG
;
```

3. Manejo de archivos:

En consola se muestra el mensaje de ingresar nombre de archivo, el cual debe de ir sin la extensión de este.

Para almacenar el contenido del archivo se crea una variable de tipo string llamada line, mediante el método **File.ReadAllText** se lee todo el contenido del archivo.

```
//abrir archivo
line = File.ReadAllText(name+".xe"); // se completa el nombre del archivo con la extension .xe
line = line.Replace("\r", String.Empty); //quitar \r del programa
```

Al tener el contenido del archivo almacenado en la variable, se sustituyen los \r de todo el contenido debido a que esto generaba un error al momento de estar haciendo el análisis.

Para crear el archivo de salida con los errores encontrados se utiliza el método **StreamWriter**, al cual se le pasa la ruta en donde se generará el nuevo archivo junto con el nombre que este tendrá.

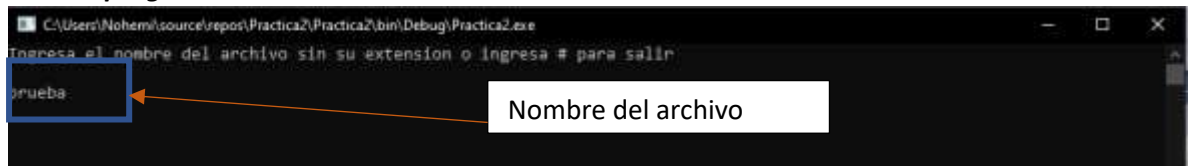
```
using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\NoheMi\source\repos\Practica2\archivoErrores.err", true))
{
    file.WriteLine("Error de lexer: " + msg + " line: " + line + " position: " + charPositionInLine);
}
}
```

Mediante **file.WriteLine** se escribe en el archivo antes creado.

4. Código

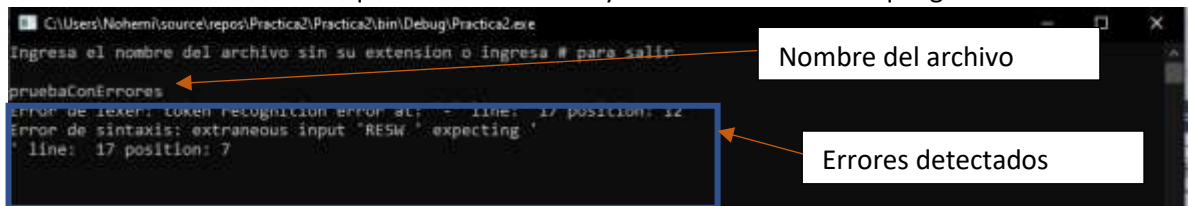
```
Console.WriteLine("Ingresa el nombre del archivo sin su extension o ingresa # para salir\n"); //línea en consola
string name = ""; //variable para almacenar la cadena de entrada que es el nombre del archivo
string line = "";
name = Console.ReadLine();
if (String.Compare(name, "#") == 1) //si no detecta un # continua con el programa
{
    //abrir archivo
    line = File.ReadAllText(name+".xe"); // se completa el nombre del archivo con la extension .xe
    line = line.Replace("\r", String.Empty); //quitar \r del programa
    sicxeLexer lex = new sicxeLexer(new AntlrInputStream(line + Environment.NewLine)); //lexer
    lex.RemoveErrorListeners(); //quitar listeners para poder usar el metodo de errores propio
    lex.AddErrorListener(ErroresUno.INSTANCE); //agregar metodo de manejo de errores
    CommonTokenStream tokens = new CommonTokenStream(lex); //crear tokens segun el lexer
    sicxeParser parser = new sicxeParser(tokens); //parser con tokens creados
    parser.RemoveErrorListeners(); //quitar listeners para poder usar el metodo de errores propio
    parser.AddErrorListener(ErroresToken.INSTANCE); //agregar metodo de manejo de errores
    try
    {
        parser.programa(); //llamada a parser en programa
    }
    catch (RecognitionException e)
    {
        Console.WriteLine(e);
    }
}
else { Environment.Exit(1); }
Console.ReadLine();
```


5. Uso del programa



Al dar click en la tecla enter, notamos que no se muestra ningún error en consola, el archivo que hemos abierto no tiene errores y por lo tanto no genera un archivo con errores.

Ahora abriremos un archivo que contiene errores y observamos la salida que genera.



Al dar click en la tecla enter, notamos que nos muestra errores, el archivo que hemos abierto tiene errores y por lo tanto si genera un archivo con errores que se encuentra en la ruta indicada en la clase de manejo de errores:

```
public class ErroresUno : IAntlrErrorListener<int>
{
    public static ErroresUno INSTANCE = new ErroresUno();

    public void SyntaxError([NotNull] IRecognizer recognizer, [Nullable] int offendingSymbol, int line, int charPositionInLine, [NotNull] string msg, [Nullab]
    {
        String sourceName = recognizer.InputStream.SourceName;

        Console.WriteLine("Error de lexer: " + msg + " line: " + line + " position: " + charPositionInLine);

        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\Nohemi\source\repos\Practica2\archivoErrores.err", true))
        {
            file.WriteLine("Error de lexer: " + msg + " line: " + line + " position: " + charPositionInLine);
        }
    }
}

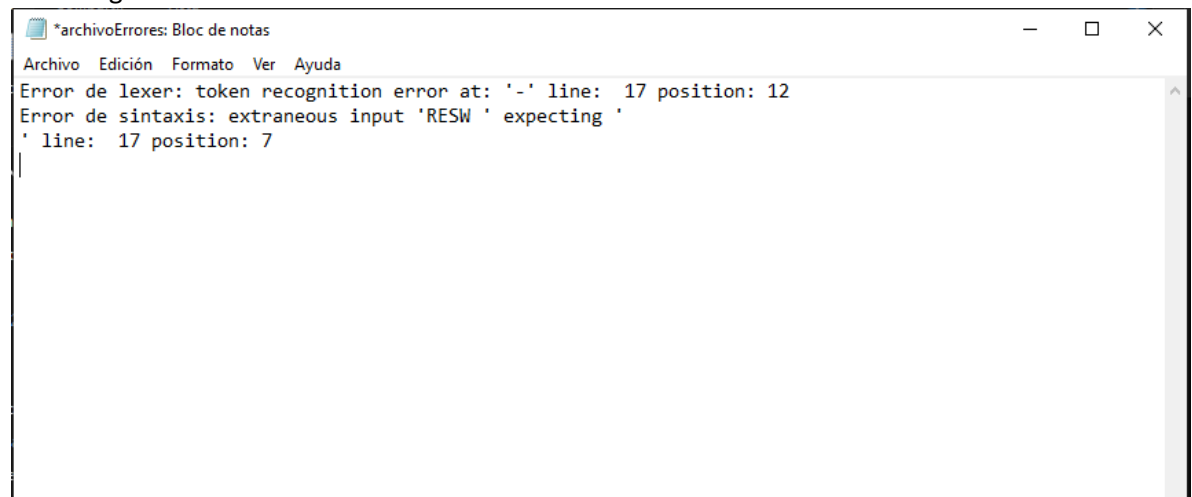
public class ErroresToken : IAntlrErrorListener<IToken>
{
    public static ErroresToken INSTANCE = new ErroresToken();

    public void SyntaxError([NotNull] IRecognizer recognizer, [Nullable] IToken offendingSymbol, int line, int charPositionInLine
    {
        Console.WriteLine("Error de sintaxis: " + msg + " line: " + line + " position: " + charPositionInLine);

        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\Nohemi\source\repos\Practica2\archivoErrores.
        {
            file.WriteLine("Error de sintaxis: " + msg + " line: " + line + " position: " + charPositionInLine);
        }
    }
}
```

Para el manejo de errores se agregaron dos clases que implementan/extienden de `IAntlrErrorListener`.

Archivo generado:



```
*archivoErrores: Bloc de notas
Archivo Edición Formato Ver Ayuda
Error de lexer: token recognition error at: '-' line: 17 position: 12
Error de sintaxis: extraneous input 'RESW ' expecting '
' line: 17 position: 7
```

6. Problemas durante el desarrollo de la practica

Los problemas que tuve fueron al momento de estar generando la gramática, ya que por el enter (\r\n) que tenían mis archivos para pruebas generaban errores en líneas que estaban correctas. También tuve problemas con la recursividad en la regla de proposiciones y con el manejo de los espacios.

7. Conclusiones y posibles mejoras

En esta práctica recordé conceptos que ya había visto en la materia de compiladores, sin embargo, y otros mas que no recordaba los tuve que investigar para poder corregir los errores que tuve al momento de desarrollar mi gramática.

Me ayudo mucho estar investigando acerca del manejo de errores en el Antlr debido a que encontré en el camino cosas interesante sobre el antlr que me ayudaron a comprender mas el como es que se tienen que generar las gramáticas.

NOTA: los archivos con los que se probó el programa están el carpeta Practica2\bin\Debug, con los nombres

- prueba.xe
- pruebaConErrores.xe