PRACTICA 2
ANALIZADOR LEXICO SINTACTICO

Laboratorio de Programación de sistemas

Oscar Armando González Patiño

Componentes léxicos definidos

Llamada del programa principal, se compone de la sentencia de inicio, un numero de proposiciones y la línea final:

```
prog:
	start|propositions|end
;
```

Estructura básica de una sentencia:

```
start:
	label START NUM ENDL|proposition
;
```

Indica la oración final que se tiene que escribir:

```
end:
	END input ENDL|END input
;
```

Define si la directiva de END tiene una etiqueta o no:

```
input:
	LABEL?
;
```

Estructura de una instrucción en general:

```
propositions:
	propositions proposition|proposition
;
```

Estructura de una instrucción o una directiva:

```
proposition:
	instruction|directive
;
```

Estructura de una instrucción. En los argumentos puede llevar una etiqueta direccionada o indexada:

```
instruction:
	label INSTR instr_args ENDL
;
```

Estructura de la directiva:

```
directive:
	label DIRECTIVE directive_args ENDL
;
```

Define si existe o no la etiqueta para una instrucción:

```
label:
	LABEL?
;
```

Estructura de una etiqueta, que puede ser direccionada o indexada, o no tenerla:

```
instr_args:
	LABEL', X'|LABEL',X'|LABEL?
;
```

Estructura del argumento para una directiva: indica si es hexadecimal o cadena:

```
directive_args:
	NUM|'X\'' NUM '\''|'C\'' LABEL '\''
;
```

Reglas gramaticales utilizadas

Palabras reservadas para los códigos de operación:

```
INSTR:		'ADD'|'AND'|'COMP'|'DIV'|'J'|'JEQ'|'JGT'|'JLT'|'JSUB'|'LDA'|'LDCH'|'LDL'|'LDX'
		|'MUL'|'OR'|'RD'|'RSUB'|'STA'|'STCH'
		|'STL'|'STSW'|'STX'|'SUB'|'TD'|'TIX'|'WD';
```

Palabras reservadas para las directivas:

```
DIRECTIVE:	'BYTE'|'WORD'|'RESB'|'RESW';
```

Palabra reservada para start y end:

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

Define una etiqueta de por lo menos un carácter:

```
LABEL:		[A-Z]+;
```

Define un numero hexadecimal:

```
NUM:		NUM_CHAR+('h'|'H')?;
NUM_CHAR:	[a-f]|[A-F]|[0-9];
```

Define una línea vacía y saltos de línea:

```
EMPTY:		' ';
ENDL:		'\n';
WS:			(' '|'\r'|'\n'|'\t') -> channel(HIDDEN);
```

Manejo de archivos de entrada y salida

Entrada

Para abrir un archivo primero se define que solo se puedan seleccionar archivos de extensión (“.s”) y se abre una ventana donde seleccionas el archivo que se desea abrir. Una vez que se abre el archivo, se aplica el método ReadAllLines al archivo para obtener las líneas de código, llevando como parámetro únicamente la ruta del archivo del cuál se van a extraer, y asignarlas en el TextBox donde se podrá editar el programa.

```
OpenFileDialog open = new OpenFileDialog
{
    InitialDirectory = Application.StartupPath + "\\example",
    Filter = "SIC File (*.s)|*.s|All files (*.*)|*.*",
    DefaultExt = ".s"
};
if (open.ShowDialog() == DialogResult.OK)
{
    tbPrograma.Lines = File.ReadAllLines(open.FileName);
    string[] files = open.FileName.Split((char)92);
    string[] file = files[files.Length - 1].Split('.');
    nombre = file[0];
    ActualizaNumeroLineas();
    ActiveForm.Text = "SIC - " + nombre;
}
```

Salida

Se usaron dos maneras de guardar el programa editado, “guardar” y “guardar como”. En “guardar” la forma que se usa es sobrescribir el archivo existente en la ruta guardada con anterioridad con las nuevas líneas de código. Se usa el método WriteAllLines, llevando como parámetros la ruta del archivo en el cual se van a escribir las líneas de código, y el código a escribir.

```
if (ruta != null)
{
    File.WriteAllLines(ruta + @"\" + nombre + ".s", tbPrograma.Lines);
    tbPrograma.DeselectAll();
}
```

La función “guardar como” abre una ventana donde seleccionas la ubicación donde vas a guardar el archivo y le asignas un nombre, agregando la extensión (“.s”) automáticamente. Después, al archivo creado recientemente se usa el mismo

método que en “guardar”, se implementa el método WriteAllLines, llevando como parámetros de igual manera la ruta del archivo, y el código a guardar.

```
SaveFileDialog save = new SaveFileDialog
{
    InitialDirectory = Application.StartupPath + "\\example",
    Filter = "SIC File (*.s)|*.s|All files (*.*)|*.*",
    DefaultExt = ".s"
};
if (save.ShowDialog() == DialogResult.OK)
{
    File.WriteAllLines(save.FileName, tbPrograma.Lines);
}
```

Conclusiones

En esta práctica se implemento una aplicación para realizar el análisis léxico y sintáctico para el lenguaje ensamblador de la arquitectura SIC en su versión estándar. Se muestran los errores del programa analizado y se crea un archivo con las líneas que los contienen. Se hace uso del manejo de archivos de extensión (“.s”) pudiendo abrirlos, modificarlos y guardarlos para su posterior uso.