Índice

Introducción

Esta práctica está integrada por dos tipos de generadores por una parte tenemos a Lex:

Lex nos ayuda a escribir programas cuyo control de flujo está dirigido por instancias de expresiones regulares en el flujo de entrada. La fuente de Lex es una tabla de expresiones regulares y fragmentos de programas correspondientes, dicha tabla es traducida a un programa que lee el flujo de entrada, copiándola en una secuencia de salida y divide la entrada en cadenas que coinciden con las expresiones dadas.

Y también hacemos uso de Yacc, que nos proporciona una herramienta para describirle la entrada a la computadora. El programador específica la estructura de la entrada junto con el código necesario que se va a invocar cuando cada estructura es reconocida.

Junto con estas dos herramientas crearemos un analizador de “lenguaje natural” de tipo SVO (Sujeto, Verbo, Objeto) y la transforma a OSV la estructura del lenguaje del personaje Yoda de la guerra de las galaxias.

Desarrollo

Para comenzar debemos de tener en cuenta cómo se estructura una frase SVO normalmente. Para ello usaremos la notación Backus-Naur:

<frase> ::= <sujeto> <verbo> <objeto>

<sujeto> ::= <artículo> <sustantivo> | La persona

<artículo> ::= el | la | los | las | uno | unos | un | ...

<sustantivo> ::= gato | perro | hombre | mujer | computadora | robot

<verbo> ::= hizo | corrió | comió | programó | enseñó

<objeto> ::= <artículo> <sustantivo> | dos memorias

Lo primero que tenemos que realizar es el analizador léxico con Flex (Lex) y poner los posibles *inputs* del usuario y de nuestra gramática, lo realizamos de la siguiente manera dentro del bloque “%% %%” (La sección de las reglas/patrones):

```
“el” {yylval=0; return(ARTICULO);}
```

Donde:

“el” es la expresión regular o cadena (en este caso una cadena) que se espera se introduzca en la entrada.

{ } dentro de los corchetes se pone el código que se va a ejecutar cuando el *input* coincida con la cadena o expresión regular.

yylval en esta variable global se almacena el valor semántico del token. Nos ayuda a evaluar el token en el analizador sintáctico. Véase.

return el tipo de token de la cadena o expresión regular.

Posteriormente se hace esto secuencialmente con todas las reglas (cadenas o expresiones regulares) que tengamos donde ***yylval*** se irá aumentando de uno en uno.

Después de terminar con el sencillo analizador léxico seguimos con el analizador sintáctico.

Dentro del bloque de definiciones se importan las librerías C stdio.h y ctype.h, luego comienza con un array de tokens ordenados por el valor en ***yylval*** que tienen los tokens en el analizador léxico.

No vamos a describir todo el código en sí, sólo las partes más importantes, así que nos saltamos a la línea 21 del código.

Podemos ver que la línea que decimos tiene una sintaxis distinta y es porque es sintaxis yacc. La palabra reservada ***start*** le indica al analizador cuál es el primer "comando" que debería iniciarse primero a analizar, por eso nosotros pusimos lista.

Seguimos con los tokens ARTICULO SUSTANTIVO VERBO SUJETO OBJETO que se retornaban en los tokens del analizador léxico.

Seguimos con el bloque de reglas donde empezamos con el primero en ser analizado que es "lista". "lista" tiene diferentes opciones desde la opción por defecto donde llama a la función iniciar partes después analiza dos posibles opciones, una donde el usuario ingresa el flujo de manera correcta con el orden SVO y otra de excepciones por si el usuario comete un error al ingresar dicho flujo SVO.

En la opción donde analiza una frase se derivan otras opciones, la primera analiza la opción de que el usuario ingrese el carácter "&" para salir del programa y la otra donde se tenga en cuenta la estructura SVO.

Cada opción dentro de *frase* como por ejemplo *sujeto* tiene diferentes opciones para analizar, ya sea que un *sujeto* se construya de manera *articulo sustantivo* o sólo con *sujeto*, entonces a cada uno le pertenece una regla que está enumerada simplemente por el número del renglón de las definiciones BNF.

Al finalizar el análisis de las palabras y con las palabras guardadas dentro de arrays el analizador vuelve e imprime la cadena "Yoda dice: %s %s %s" cadenaobjeto, cadenasujeto, cadenaverbo.

Al ejecutarse se escribe la frase en forma SVO y los analizadores hacen lo suyo devolviendo la frase en forma OSV.

Ejecución

```
→ lex-yacc git:(master) ✗ ./yoda
el gato ve robots
 Regla 1: el gato Regla 4: ve Regla 5: robots

Yoda dice: robots el gato ve

Siguiente frase por favor!

la alumna estudia programacion
Regla 5: la alumna Regla 4: estudia Regla 5: programacion

Yoda dice: programacion la alumna estudia

Siguiente frase por favor!

el gato programa sitios web
 Regla 1: el gato Regla 4: programa Regla 5: sitios web

Yoda dice: sitios web el gato programa

Siguiente frase por favor!

el perro hizo pastel
 Regla 1: el perro Regla 4: hizo Regla 5: pastel

Yoda dice: pastel el perro hizo

Siguiente frase por favor!
```

Conclusión

Lex y Yacc son dos herramientas muy poderosas y de alto nivel que nos permiten construir compiladores y también analizadores de lenguaje natural. Aunque este ejercicio no se acerca para nada a un analizador de lenguaje natural nos da unas bases para saber cómo se construyen dichos analizadores y el potencial que le podemos sacarles. Nos decidimos por este tipo de analizador porque nos pareció algo sencillo y entretenido de hacer en lugar de escribir un lenguaje de

programación que por nuestras habilidades en el momento de escribir todo esto nos hubiera traído más dolores de cabeza que resultados, y volvemos a reiterar, tal vez no sea el ejercicio más avanzado, pero este nos ayudó a entender más sobre las herramientas y estamos satisfechos con nuestro trabajo.

Referencias

1. **lex & yacc. John R. Levine, Tony Mason & Doug Brown. O'Reilly 1992**
2. **https://www.geeksforgeeks.org/introduction-to-yacc/ introducción práctica a yacc**