



**Universidad Nacional
Autónoma de México**
Facultad de Ingeniería



Compiladores

Profesor: Manuel Castañeda Castañeda

Semestre 2022-2

Grupo

Analizador sintáctico ascendente

Integrantes:

Nonoal Olivares, Luis Enrique

Arriaga Mejía, José Carlos

Introducción

En este proyecto se realizará un Analizador Sintáctico Ascendente (ASA) que deberá revisar las sentencias de acuerdo a la siguiente gramática

$E \rightarrow T$
 $E \rightarrow E+T$
 $E \rightarrow E-T$
 $T \rightarrow i$
 $T \rightarrow (E)$

Donde i será cualquier número real o identificador.

La cadena de tokens a revisar será entregada en un archivo por un Analizador Léxico (AL) hecho en el lenguaje de predilección

El analizador léxico reconocerá: números reales, identificadores símbolos de operación (+,-) y los paréntesis (,)

El ASA recibe el archivo con las cadenas de tokens separadas por ;

Deberá indicar si es cadena válida

Deberá indicar en qué líneas se encuentran los posibles errores

Desarrollo

Para el AL utilizaremos dos archivos .txt, el archivo prueba.txt tendrá distintas combinaciones de números reales, identificadores, paréntesis y operadores + y -, cada cadena que analizará está separada por un ;

El AL leerá éste archivo e identificará los “(”, “)”, “+”, “-” cuando encuentre alguno de estos, los escribirá como tal en el archivo AL.txt, cuando encuentre números reales o identificadores los escribirá como i , si encuentra cualquier otro carácter escribirá “e” de error que será identificado en el próxima parte.

prueba.txt	AL.txt
3- (3.1416+(3-4));	i- (i+(i-i));
Jose+(3- (8.15+(_Kike-Carlos_)));	i+(i- (i+(i-i)));
id;	i;
(b);	(i);
a+b;	i+i;
();	();
3++;	i++;
-3+3;	-i+i;
(5*5)+5;	(iei)+i;
(9)/(5);	(i)e(i);
;	;
78.255- (-84545+5)	i- (-i+i)
;	;
pP_Asa+(asdQDAsad_99-69);	i+(i-i);

```

1 %option yylineno
2 %option noyywrap
3
4 %{
5     #include <stdio.h>
6 }
7 /* Identifica numeros enteros*/
8 digito [0-9]+
9 /*Identifica numeros con decimal*/
10 real [0-9]*\.[0-9]+
11 /*Identifica cualquier letra*/
12 letra [A-Za-z]
13 identificador (_|{letra})(_|{letra}|{digito})*
14 /*Identifica identificadores validos en C*/
15
16 %%
17 {digito} {fprintf(yyout, "i");}
18 {real} {fprintf(yyout, "r");}
19 {identificador} {fprintf(yyout, "i");}
20 /*En las 3 anteriores escribe en el archivo AL.txt una i
21 cada que encuentra alguna de ellas
22 En los siguientes escribe en AL.txt el simbolo que encuentre*/
23 [\()] {fprintf(yyout, "(");}
24 [\)] {fprintf(yyout, ")");}
25 "+" {fprintf(yyout, "+");}
26 "-" {fprintf(yyout, "-");}
27 [";"] {fprintf(yyout, ";");}
28 "\n" {fprintf(yyout, "\n");}
29 . {fprintf(yyout, "e");}
30
31 %%
32 int main() {
33     FILE *in;
34     FILE *out;
35     /*Abre el archivo prueba.txt donde esta la entrada que analizara*/
36     yyin = fopen("prueba.txt", "r");
37     /*Abre el archivo AL.txt donde ira transformando la entrada en los tokens*/
38     yyout = fopen("AL.txt", "w");
39     yylex();
40     return 0;
41 }

```

Para la siguiente parte utilizaremos el archivo AL.txt que nos dio el AL. En el archivo anasin.y estará las reglas de producción mencionadas al principio del documento, (éstas reglas ya están factorizadas y sin recursividad para evitar problemas) y los tokens que recibirá del segundo AL.

```

home > arriaga > Desktop > Proyecto (1) > anasin.y
1  %{
2      #include <stdio.h>
3      #include <ctype.h>
4      extern int yylex(void);
5      void yyerror(char *mensaje);
6  }
7
8  %token I
9  %token PABRE
10 %token PCIERRA
11 %token RESTA
12 %token SUMA
13 %token PUNTOCOMA
14
15 %start INICIO
16
17 %%
18 INICIO : INICIO E PUNTOCOMA {printf("CADENA VALIDA\n");}
19       | E PUNTOCOMA {printf("CADENA VALIDA\n");}
20       | error PUNTOCOMA {yyerrok;}
21       ;
22
23 E      : T
24       | T EP
25       ;
26
27 EP     : SUMA T
28       | RESTA T
29       /*VACIO*/
30       ;
31
32 T      : I
33       | PABRE E PCIERRA
34       ;
35
36 %%
37 int main (void){
38     yyparse();
39     return 0;
40 }
41

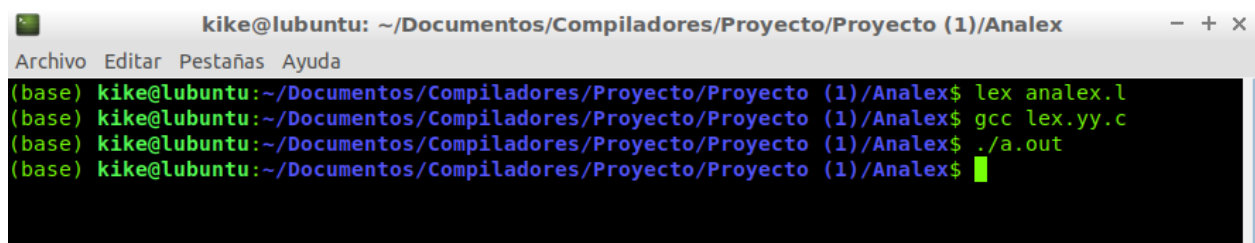
```

El segundo AL leerá el archivo entregado por el primer AL, identificará el alfabeto de la gramática y mandará el tokens correspondiente al AS que irá recorriendo las producciones de la gramática.

```
1  %{
2      #include <stdio.h>
3      #include "y.tab.h"
4      int yylex();
5      void yyerror(char *msg);
6      int lineno = 1;
7  %}
8
9  %option yylineno
10 %option noyywrap
11
12 %%
13
14 "i"          {return I;}
15 "("          {return PABRE;}
16 ")"          {return PCIERRA;}
17 "+"          {return SUMA;}
18 "-"          {return RESTA;}
19 ";"          {return PUNTOCOMA;}
20 "e"          {printf("Error lexico\n");}
21 "."          {printf("Error lexico\n");}
22 "\n"         {lineno++;}
23
24 %%
25 void yyerror(char *msg) {
26     printf("Error en la línea: %d\nLa cadena ingresada no es valida para la gramática\n", lineno);
27 }
28
29
```

Análisis de resultados

El primer análisis que se hizo fue cuando se ejecutó el analex.l junto con el lex.yy.c.



```
kike@lubuntu: ~/Documentos/Compiladores/Proyecto/Proyecto (1)/Analex
Archivo  Editar  Pestañas  Ayuda
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)/Analex$ lex analex.l
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)/Analex$ gcc lex.yy.c
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)/Analex$ ./a.out
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)/Analex$ █
```

Se ejecuta primero el analex debido a que lo que éste hace es identificar números enteros, números con decimal o cualquier letra. Cada vez que encuentra una letra, número o identificador, en un nuevo archivo de nombre AL.txt se irán reconociendo estos caracteres y se escribirá una "i". Además de eso, se escribirán los símbolos que vaya encontrando, por ejemplo los paréntesis, saltos de línea, operadores y punto y coma. Cuando se ejecuta el archivo analex lo que se está haciendo es abrir el archivo "prueba.txt" que es donde se tienen los datos

que analizará, y luego de analizar se escribe un nuevo archivo de nombre “AL.txt” en donde el analéx transforma los datos de “prueba.txt” en tokens.

prueba.txt	AL.txt
3-(3.1416+(3-4));	i-(i+(i-i));
Jose+(3-(8.15+(_Kike-Carlos_)));	i+(i-(i+(i-i)));
id;	i;
(b);	(i);
a+b;	i+i;
();	();
3++;	i++;
-3+3;	-i+i;
(5*5)+5;	(iei)+i;
(9)/(5);	(i)e(i);
;	;
78.255-(-84545+5)	i-(-i+i)
;	;
pP_Asa+(asdQDAsad_99-69);	i+(i-i);

Aquí podemos ver como los números reales e identificadores los pone en el archivo AL.txt como i, los demás elementos de la gramática los pasa tal cual y cuando encuentra alguno que no pertenece a ella escribe una e.

```
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)/Analex$ cd ..
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)$ yacc -d anasin.y
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)$ lex alex.l
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)$ gcc lex.yy.c y.tab.c -lfl
(base) kike@lubuntu:~/Documentos/Compiladores/Proyecto/Proyecto (1)$ ./a.out
```

Luego ejecutamos con yacc el archivo “anasin.y” y también el archivo “alex.l”. Cuando se ejecuta el yacc con la instrucción -d lo que se hace es generar un nuevo archivo de nombre “y.tab.h” que necesita el “alex.l”.

Cuando se ejecuta el “lex.yy.c y.tab.c -lfl”, con el lfl lo que se hace es cargar las librerías del yacc para que corra correctamente.

Finalmente, con el comando “./a.out <AL.txt” lo que se hace es ejecutar y analizar el archivo creado anteriormente (AL.txt) en donde obtendremos como resultado si la cadena es válida o no, además de obtener la línea en donde se encuentra el error.

```
AL.txt x
i-(i+(i-i));
i+(i-(i+(i-i)));
i;
(i);
i+i;
();
i++;
-i+i;
(iei)+i;
(i)e(i);
;
i-(-i+i)
;
i+(i-i);|
```

```
arriaga@UnTalArriaga:~/Desktop/Proyecto (1)$ ./a.out <AL.txt
CADENA VALIDA
CADENA VALIDA
CADENA VALIDA
CADENA VALIDA
CADENA VALIDA
Error en la linea: 6
La cadena ingresada no es valida para la gramática
Error en la linea: 7
La cadena ingresada no es valida para la gramática
Error en la linea: 8
La cadena ingresada no es valida para la gramática
Error lexico en la linea 9
Error en la linea: 9
La cadena ingresada no es valida para la gramática
Error lexico en la linea 10
Error en la linea: 10
La cadena ingresada no es valida para la gramática
Error en la linea: 11
La cadena ingresada no es valida para la gramática
Error en la linea: 12
La cadena ingresada no es valida para la gramática
CADENA VALIDA
arriaga@UnTalArriaga:~/Desktop/Proyecto (1)$
```

Podemos ver cómo es que el segundo AL detecta la “e” del primer AL y nos marca el error léxico y en qué línea ocurrió. De la misma manera cuando la cadena no es válida en nuestra gramática nos marca que no es válida y en qué línea está el error.

Conclusiones

Arriaga Mejía José Carlos.

La creación de un ASA con ayuda de YACC es mucho más sencillo que si lo hiciéramos en C, ya que YACC lo pasa a C él solo. A pesar de eso tuvimos varios errores durante la elaboración de ASA, por ejemplo que al principio solo leía una línea del archivo y ahí quedaba el ASA que se arregló agregando una producción donde se iniciaría y agregando en esta una recursividad, también que después del primer error encontraba todo lo demás lo marcaba como error, además había cadenas las cuales daba como válidas cuando no debía, esto se arregló factorizando y quitando la recursividad de la producción E. En conclusión, no basta que YACC nos ayude en la mayoría de los procedimientos, es necesario saber de tema para corregir los errores que no puede corregir YACC, que son los errores de corrección.

Nonoal Olivares Luis Enrique

Pudimos concluir satisfactoriamente este proyecto el cual era desarrollar un analizador sintáctico ascendente. A lo largo de este proyecto pude darme cuenta de distintas cosas, por ejemplo, que yacc es mucho más fácil de utilizar el lenguaje C, aunque por su puesto que hay que tener conocimiento en yacc ya que si no lo hay no se podrían corregir los errores.

Para este proyecto se puso en práctica todo lo visto en la teoría, desde un análisis léxico hasta un análisis sintáctico ascendente, en el cual sí tuvimos un poco de problemas al inicio debido a problemas de lectura en cuestión del archivo de prueba, pero finalmente se pudo corregir y con ello el proyecto se completó satisfactoriamente.