Diseño de compiladores

Tarea de lex y YACC

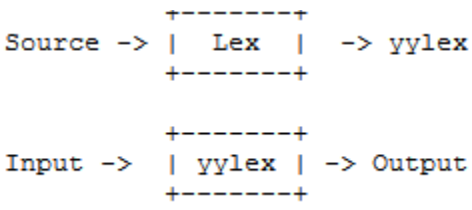
Oscar Javier Hinojosa Luna A01223081

LEX

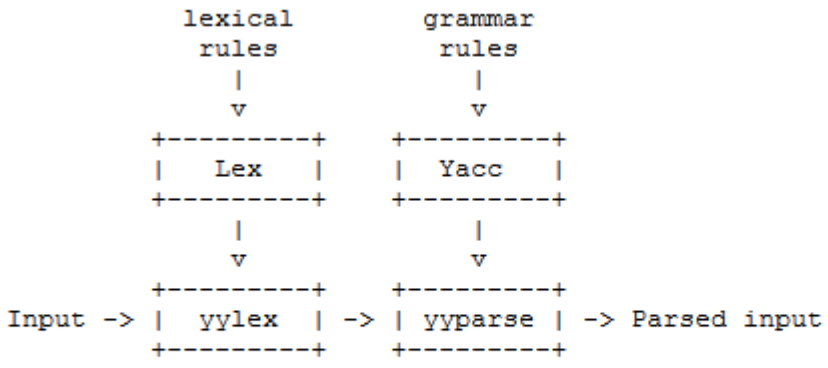
Lex es una herrarmienta que ayuda a escribir programas los cuales su dlujo esta dirijido por expresiones regulares. Esta herramienta es utilizada para el analizis sintactico. Este programa esta orientado a problemas de alto nivel para el reconocimiento de cadenas de caracteres y este produce una salida en un lenguaje general que reconoce las expresiones regulares, las cuales son especificadas por el programador en la secuencia de entrada y las coincidencias de la secuencia de entrada. Lex asocia las expresiones regulares y los fragmentos del programa y dependiendo de la expresión que aparezca en la entrada se ejecuta una parte del programa escrito.

Lex es un generador para le representacion de una nueva parte de un lenguaje que peude ser agregado diferentes lenguajes. Además, genera código que puede ejecutarse en diferentes HWs, adicionalemente permite la generación de código de diferentes lenguajes.

Lex convierte las expresiones y acciones en el lenguaje de uso general del host el programa generado se llama yylex. El programa yylex reconocerá expresiones en una secuencia (llamada entrada en esta nota) y realizará las acciones especificadas para cada expresión a medida que se detecte.



Lex se puede usar solo para transformaciones simples, o para análisis y recopilación de estadísticas en un nivel léxico. Lex también se puede usar con un generador de analizador para realizar la fase de análisis léxico; es particularmente fácil conectar Lex y Yacc [3]. Los programas de Lex reconocen solo expresiones regulares. Cuando se usa como un preprocesador para un generador de analizador posterior, Lex se usa para dividir el flujo de entrada, y el generador del analizador asigna la estructura a las piezas resultantes. El flujo de control en tal caso. Programas adicionales, escritos por otros generadores o en forma manual, se pueden agregar fácilmente a programas escritos por Lex.



Lex with Yacc
Figure 2

Esta es la forma general de un código LEX:

```
{definitions}
%%
{rules}
%%
{user subroutines}
```

Ejemplo de LEX

```
%{
#include <stdio.h>
%}

%%
[a-zA-Z][a-zA-Z0-9]*    printf("WORD ");
[a-zA-Z0-9\/.-]+        printf("FILENAME ");
\"                      printf("QUOTE ");
\{                      printf("OBRACE ");
\}                      printf("EBRACE ");
;                       printf("SEMICOLON ");
\n                      printf("\n");
[ \t]+                  /* ignore whitespace */;
%%
```

La salida de esto es

```
WORD OBRACE
WORD FILENAME OBRACE WORD SEMICOLON EBRACE SEMICOLON
WORD WORD OBRACE WORD SEMICOLON EBRACE SEMICOLON
EBRACE SEMICOLON

WORD QUOTE FILENAME QUOTE OBRACE
```

```
WORD WORD SEMICOLON
WORD QUOTE FILENAME QUOTE SEMICOLON
EBRACE SEMICOLON
```

YACC

Yacc provee una herramienta general para analizar de manera estructurada una entrada. La usar yacc se bebe generar tres cosas una seri de reglas para describir la entrada, un código que debe ejecutarse al detectar que una regla se cumple y lo último es tener un analizador de la entrada. Después de esto el programa convierte estos en una función en un lenguaje de programación que examinara lo cadena de caracteres que ledes de entrada. Para esto primeramente debes correr LEX o alguno otro analizador léxico para obtener solo lo tokens de la cadena de entrada para poder trabajar con esto y las reglas gramaticales necesarias. Esto para poder reconocer las reglas y esto generará una acción que será una función de código que será ejecutada en ese momento la cual puede retornar valores y usar los valores retornados por otras acciones

Un programa fuente de Yacc se parece bastante a uno de lex. La diferencia principal está en la sección de reglas, que en vez de expresiones regulares contiene las reglas de la gramática:

```
<sección de definiciones>
%%
<sección de reglas>
%%
<sección de rutinas>
```

Ejemplo de YACC

```
%{
#include <stdio.h>
%}

%token NAME NUMBER
%%
statement:     NAME '=' expression    { printf("pretending to assign %s
the value %d\n", $1, $3); }
        /* Note that in the current version, the lexer never actually
returns NAME, so this rule will never happen. */
        ;
        |      expression             { printf("= %d\n", $1); }
        ;
expression:    expression '+' NUMBER  { $$ = $1 + $3;
                                        printf ("Recognized '+'
expression.\n");
                                        }
        |      expression '-' NUMBER  { $$ = $1 - $3;
                                        printf ("Recognized '-'
expression.\n");
```

```
                                        }
        |       NUMBER                  { $$ = $1;
                                        printf ("Recognized a number.\n");
                                        }
        ;
%%
int main (void) {
        return yyparse();
        }

/* Added because panther doesn't have liby.a installed. */
int yyerror (char *msg) {
        return fprintf (stderr, "YACC: %s\n", msg);
        }
```