Actividad 6

Nombre: Ricardo Jesús Leguizamón Acosta

1. ¿Aceptará la calculadora una línea que contenga solo un comentario? Por qué no? ¿Sería más fácil solucionar esto en el escáner o en el analizador?

No aceptara una línea que contenga comentario, la calculadora informa *sintax error* si ingresa una línea que contiene solo un comentario.

Sería más fácil solucionar con el analizador, ya que con el analizador podemos agregar código como regla para que no lance errores si se ingresa comentario de input.

```
calclist:
  | calclist exp EOL { printf("decimal = %d\nhexadecimal = 0x%x\n", $2,$2); }
  | calclist EOL {/*NO HACE NADA*/}
  ;
```

2. Convierte la calculadora en una calculadora hexadecimal que acepte números hexadecimales y decimales. En el escáner, agregue un patrón como: 0x[a-f0-9]+ para que coincida con un número hexadecimal, y en el código de acción use strtol para convertir la cadena en un número almacenado en yylval; luego devuelve un NUMBER token. Ajuste la salida printf para imprimir el resultado en decimal y hexadecimal.

Sintáctico:

me > rick > Desktop > Ejercicio 2 > ejer2.y

```
1   %{
2     #include <stdio.h>
3     int yylex();
4     void yyerror(char *s);
5   %}
6   %token NUMBER
7   %token ADD SUB MUL DIV ABS
8   %token EOL
9   %%
10  calclist:
11    | calclist exp EOL { printf("decimal = %d\nhexadecimal = 0x%x\n", $2,$2); }
12    ;
13  exp: factor
14    | exp ADD factor { $$ = $1 + $3; }
15    | exp SUB factor { $$ = $1 - $3; }
16    ;
17  factor: term
18    | factor MUL term { $$ = $1 * $3; }
19    | factor DIV term { $$ = $1 / $3; }
20    ;
21  term: NUMBER
22    | ABS term    { $$ = $2 >= 0? $2 : - $2; }
23  ;
24  %%
25  void main(int argc, char **argv){
26    yyparse();
27  }
28  void yyerror(char *s){
29    fprintf(stderr, "error: %s\n", s);
30  }
```

Escáner:

```
ome > rick > Desktop > Ejercicio 2 > ≡ ejer2.l
%{
# include "ejer2.tab.h"
int yyval;
%}
%%
"+"     { return ADD; }
"-"     { return SUB; }
"*"     { return MUL; }
"/"     { return DIV; }
"|"     { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n      { return EOL; }
[ \t]   { }
0x[a-f0-9]+ { sscanf(yytext,"%d\n",&yylval) ; return NUMBER; }
.       { }
%%
```

3. Agregue operadores de bits como AND y OR a la calculadora. El operador evidente que se debe usar para OR es una barra vertical, pero este ya es el operador de valor absoluto unario. ¿Qué sucede si también se lo utiliza como un operador OR binario, por ejemplo, exp ABS factor?

No sucederá nada, ya que la prioridad que se le dé al operador depende de la cantidad de operandos que esté involucrado

"...In the first expression, the bitwise-AND operator (&) has higher precedence than the logical-OR operator (||), so a & b forms the first operand of the logical-OR operation.

In the second expression, the logical-OR operator (||) has higher precedence than the simple-assignment operator (=), so b || c is grouped as the right-hand operand in the assignment. Note that the value assigned to a is either 0 or 1."

Fuente:
https://docs.microsoft.com/en-us/cpp/c-language/precedence-and-order-of-evaluation?view=msvc-160#:~:text=Examples&text=In%20the%20first%20expression%2C%20the,of%20the%20logical%2DOR%20operation.&text=The%20logical%2DAND%20operator%20

Sintáctico:

```
ome > rick > Desktop > Ejercicio3 > ejer3.y
%{
#include <stdio.h>
int yylex();
void yyerror(char *s);
%}
%token NUMBER
%token ADD SUB MUL DIV ABS CP OR AND OP
%token EOL
%%
calclist:
 | calclist exp EOL { printf("decimal = %d\nhexadecimal = 0x%x\n", $2,$2); }
  /* | calclist EOL {/*NO HACE NADA*/} */
 ;
factor: ter { $$ = $1; }
     | factor MUL ter { $$ = $1 * $3; }
     | factor DIV ter { $$ = $1 / $3; }
     ;
ter:  term
     | ter AND term {$$ = $1 & $3; }
     | ter OR term { $$ = $1 | $3; }
     ;
term: NUMBER { $$ = $1; }
     | OR term { $$ = $2 >= 0 ? $2 : -$2; }
     | OP exp CP { $$ = $2; }
     ;
exp: factor
     | exp ADD factor { $$ = $1 + $3; }
     | exp SUB factor { $$ = $1 - $3; }
     ;

%%
void main(int argc, char **argv)
{
  yyparse();
}

void yyerror(char *s)
{
  fprintf(stderr, "error: %s\n", s);
}
```

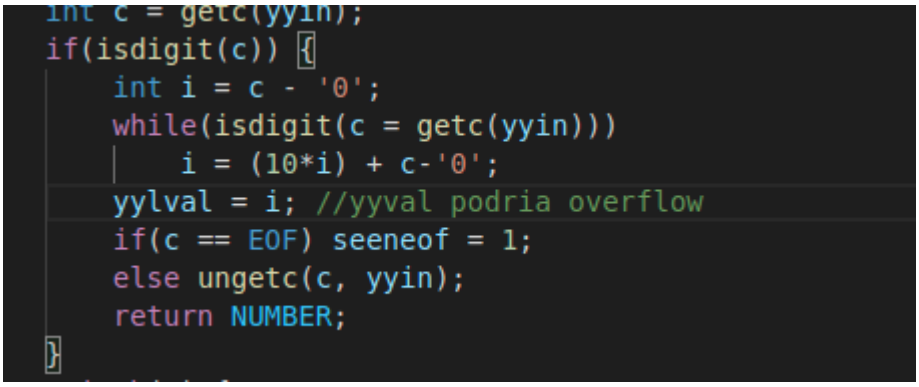
Escáner:

```
ome > rick > Desktop > Ejercicio3 > ejer3.l
%{
# include "ejer3.tab.h"
int yyval;
%}
%%
"&"     { return AND; }
"|"     { return OR; }
"+"     { return ADD; }
"-"     { return SUB; }
"*"     { return MUL; }
"/"     { return DIV; }
"|"     { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n      { return EOL; }
0x[a-f0-9]+ { sscanf(yytext,"%d\n",&yylval) ; return NUMBER; }
.       { }
%%

```

4. ¿La versión manuscrita del escáner del ejemplo 1-4 (Capítulo 1, pág.38) reconoce exactamente los mismos tokens que la versión flexible?

Si, reconocería los mismos tokens



```
int c = getc(yyin);
if(isdigit(c)) {
    int i = c - '0';
    while(isdigit(c = getc(yyin)))
        i = (10*i) + c-'0';
    yylval = i; //yyval podria overflow
    if(c == EOF) seeneof = 1;
    else ungetc(c, yyin);
    return NUMBER;
}
```

En el handwriter yylval podria darse el desbordeamiento.

5. ¿Puedes pensar en lenguajes para los que flex no sería una buena herramienta para escribir un escáner?

No sería útil para lenguajes como por ejemplo Typescript o algun POO que son muy ambiguos