**Expresiones Regulares:**

Palabras reservadas: func|end|mod|loop|cond

Operadores: [\+\-\\*\/\\^]

Identificadores: [a-zA-Z][a-zA-Z0-9]\*

Enteros: [0-9]+L?

Reales: [-+]?([0-9]\*[.][0-9]+|[0-9]+[.]?)([eE][-+]?[0-9]+)?

**Autómata Finito (Números enteros):**

A diagram of a diagram

Description automatically generated

**Autómata Finito (Numero reales):**

A diagram of a flowchart

Description automatically generated

**Implementación desde 0 del Analizador léxico:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAXLEXEMA *256*

*// #define COMP 256*

*// #define ID 257*

*// #define IF 258*

*// #define ELSE 259*

#define NUM *256*

#define REAL *257*

struct Handler

{

    FILE \**file*;

    char *lexem*[MAXLEXEMA];

};

int lex(struct Handler \**h*)

{

    size\_t *i* = *0*;

*h*->*lexem*[*i*] = fgetc(*h*->*file*);

    if (*h*->*lexem*[*i*] >= '0' && *h*->*lexem*[*i*] <= '9')

    {

        do

        {

*h*->*lexem*[++*i*] = fgetc(*h*->*file*);

        } while (*h*->*lexem*[*i*] >= '0' && *h*->*lexem*[*i*] <= '9');

        if (*h*->*lexem*[*i*] == '.')

        {

            do

            {

*h*->*lexem*[++*i*] = fgetc(*h*->*file*);

            } while (*h*->*lexem*[*i*] >= '0' && *h*->*lexem*[*i*] <= '9');

            if (*h*->*lexem*[*i*] == 'e' || *h*->*lexem*[*i*] == 'E')

            {

                do

                {

*h*->*lexem*[++*i*] = fgetc(*h*->*file*);

                } while ((*h*->*lexem*[*i*] >= '0' && *h*->*lexem*[*i*] <= '9') || (*h*->*lexem*[*i*] == '+' || *h*->*lexem*[*i*] == '-'));

*h*->*lexem*[*i*] = '*\0*';

                ungetc(*h*->*lexem*[*i*], *h*->*file*);

                return REAL;

            }

*h*->*lexem*[*i*] = '*\0*';

            ungetc(*h*->*lexem*[*i*], *h*->*file*);

            return REAL;

        }

        if (*h*->*lexem*[*i*] == 'e' || *h*->*lexem*[*i*] == 'E')

        {

            do

            {

*h*->*lexem*[++*i*] = fgetc(*h*->*file*);

            } while ((*h*->*lexem*[*i*] >= '0' && *h*->*lexem*[*i*] <= '9') || (*h*->*lexem*[*i*] == '+' || *h*->*lexem*[*i*] == '-'));

*h*->*lexem*[*i*] = '*\0*';

            ungetc(*h*->*lexem*[*i*], *h*->*file*);

            return REAL;

        }

*h*->*lexem*[*i*] = '*\0*';

        ungetc(*h*->*lexem*[*i*], *h*->*file*);

        return NUM;

    }

    return *h*->*lexem*[*i*] != EOF ? lex(*h*) : EOF;

}

int main(int *argc*, char \**argv*[])

{

    if (*argc* < *2*)

    {

        printf("Usage: *%s* <sourcefile>*\n*", *argv*[*0*]);

        return -*1*;

    }

    struct Handler \**handler* = (struct Handler \*)malloc(sizeof(struct Handler));

*handler*->*file* = fopen(*argv*[*1*], "r");

    int *tok*;

    while ((*tok* = lex(*handler*)) != EOF)

    {

        printf("Lexem (*%d*): *%s\n*", *tok*, *handler*->*lexem*);

    }

    fclose(*handler*->*file*);

    free(*handler*);

    return *0*;

}

**LEXER ANTLR:**

*lexer* *grammar* ExprLexer;

PLUS : '+' ;

MINU : '-' ;

MULT : '\*' ;

DIVI : '/' ;

BACK : '*\\*' ;

EXP  : '^' ;

FUNC: 'func';

END: 'end';

LOOP: 'mod';

COND: 'cond';

REAL : [-+]?([0-9]\*[.][0-9]+|[0-9]+[.]?)([eE][-+]?[0-9]+)*?*;

NUM: [0-9]+[L]?;

ID: [a-zA-Z\_][a-zA-Z\_0-9]\* ;

WS: [ *\t\n\r\f*]+ -> skip ;

**PARSER ANTLR**

*parser* *grammar* ExprParser;

*options* { *tokenVocab*=ExprLexer; }

program

    : lststat EOF

    ;

lststat

    : lststat stat

    |

    ;

stat

    : fn

    | loop

    | expr

    ;

fn: 'func' expr 'end';

loop: 'mod' expr 'end';

expr

    : expr expr ('\*'|'/'|'+'|'-')

    | NUM

    | REAL

    ;

**ARBOLES AST :**

AST:  
  
Ejemplo 1: Función (fn 12 12 +)

A diagram of a function

Description automatically generated

Ejemplo 2: Loop (mod 10e12 + 12 end)

A diagram of a computer program

Description automatically generated with medium confidence