Forrás: `http://cs.bme.hu/ terek/algel/lexyacc.tgz`
Az bemenetet nem karakterenmként, hanem nagyobb darabokban (token) dolgozza fel az elemző, ezért azt először ,,tokenizáljuk". A `lex` program az alábbi (`lexer.l`) bemenetből generál egy `.c` programot, ami ezt elvégzi.

```
%{
#define YYSTYPE double
#include "parser.h"
extern YYSTYPE yylval;
%}

%option noyywrap

%%

\.[0-9]+    |
[0-9]+(\.[0-9]*)?([eE][+-]?[0-9]+)? { yylval = atof (yytext); return NUMBER; }
[-+*/^()=]     { return *yytext; }
[ \t\n]        ;
.              { yyerror("Unknown character.");}

%%

int yyerror(char *msg)
{
   printf("%s\n", msg);
}
```

lexer.l

Az eredmény a `lexer.c`, ami egy `int yylex()` függvényt definiál. Ennek egy hívása visszaadja a következő token típusát (az értékét pedig egy globális változóban).

Az elemző (`parser.y`) a tokeneket kapja bemenetként, és LR(1)-gyel elemzi:

```
%{
#define YYSTYPE double
#include "parser.h"
#include <math.h>
#define  YYTRACE
%}

%token NUMBER
%start kif

%%

kif   :
   | kif expr '='    { printf ("%.16g\n", $2); $$=$2; }
   ;

expr  : term         { $$ = $1; }
   | expr '+' term   { $$ = $1 + $3; }
   | expr '-' term   { $$ = $1 - $3; }
   ;

term  : fact         { $$ = $1; }
   | term '*' fact   { $$ = $1 * $3; }
   | term '/' fact   { $$ = $1 / $3; }
   ;

fact  : num          { $$ = $1; }
   | fact '^' num    { $$ = pow ($1,$3); }
   ;

num   : val       { $$ = $1; }
   | '-' num      { $$ = -$2; }
   | '+' num      { $$ = $2; }
   ;

val   : NUMBER       { $$ = $1; }
   | '(' expr ')'    { $$ = $2; }
   ;
%%
```

parser.y