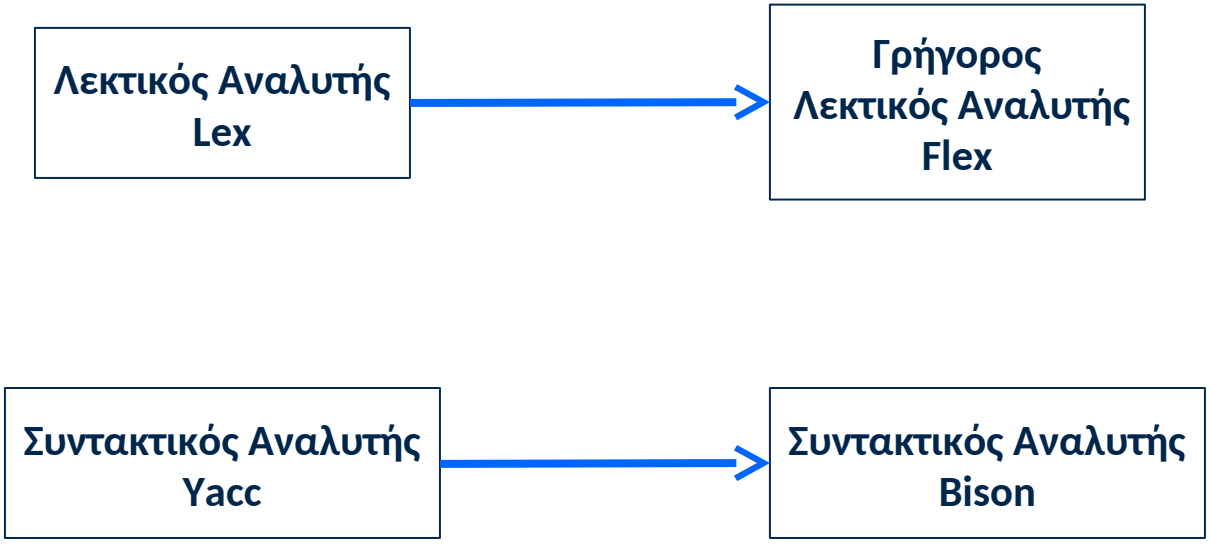
Λεκτική ανάλυση με το εργαλείο Lex

Διαλέξεις στο μάθημα: Μεταφραστές ΙΙ

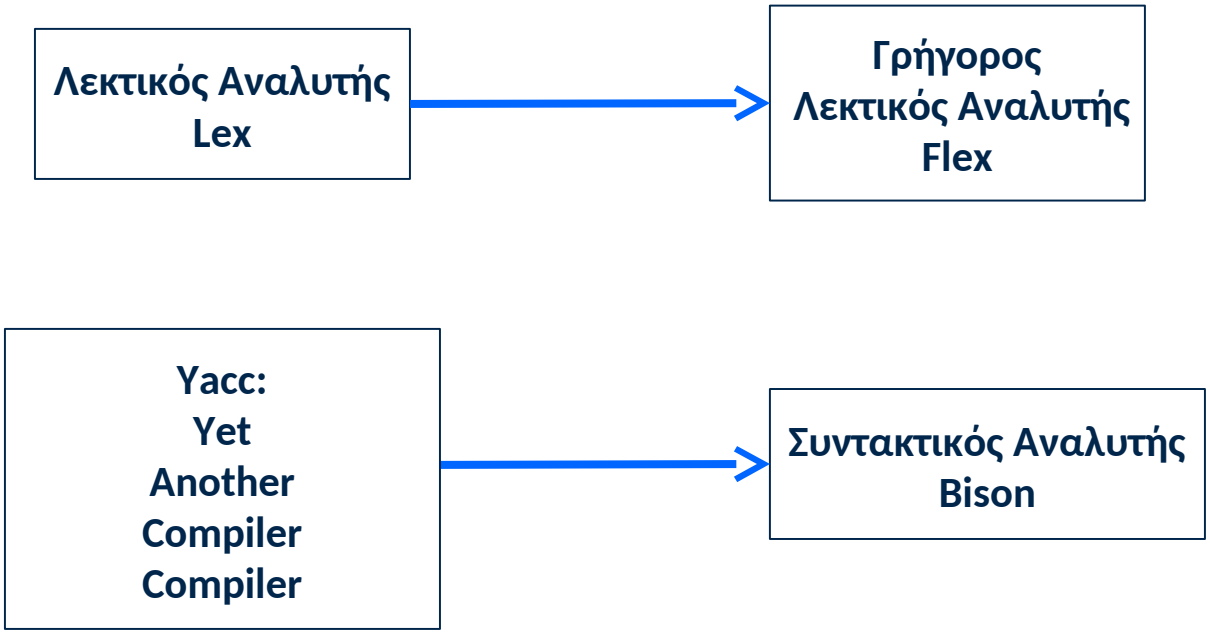
Γιώργος Μανής

Lex και Flex, Yacc και Bison

Lex και Flex, Yacc και Bison



Lex και Flex, Yacc και Bison



Lex και Flex, Yacc και Bison

Λεκτικός Αναλυτής
Lex

→ Γρήγορος Λεκτικός Αναλυτής Flex

Yacc:
Another
Compiler
Compiler

→ Συντακτικός Αναλυτής

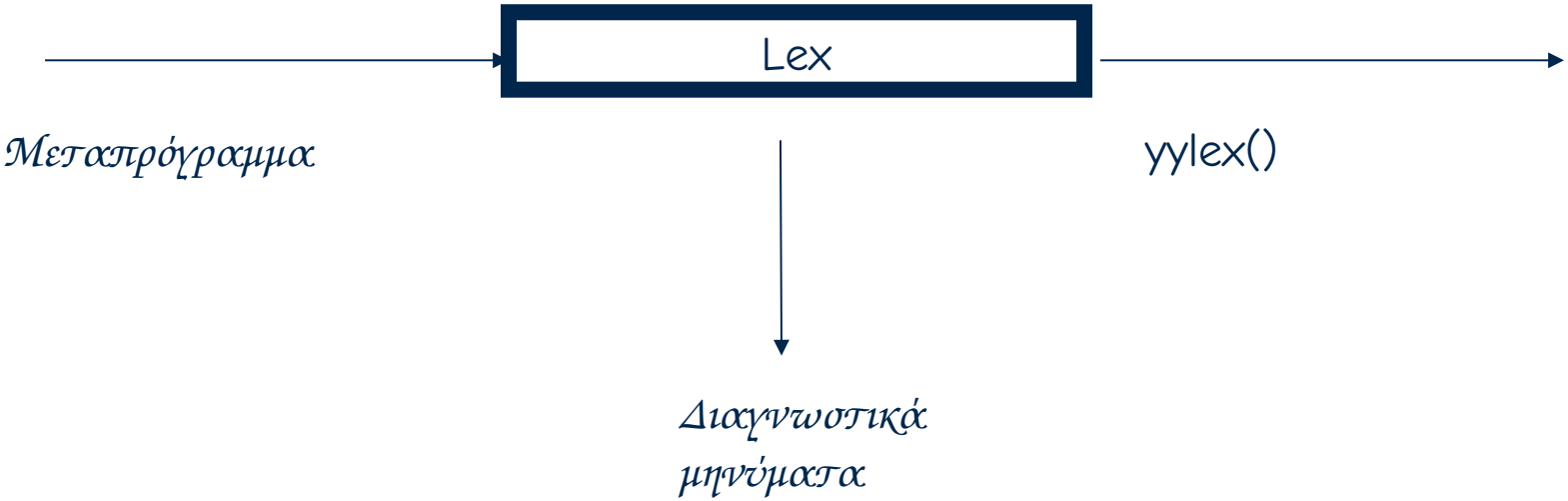
Γενικά

- το μεταεργαλείο lex (flex) είναι **ένας γεννήτορας λεκτικών αναλυτών**
- δέχεται σαν είσοδο ένα **μεταπρόγραμμα** που περιγράφει τις προς αναγνώριση λεκτικές μονάδες καθώς και τις ενέργειες που πρέπει να γίνουν όταν αυτές αναγνωριστούν
- η έξοδος είναι **ένα πρόγραμμα σε γλώσσα C** που περιέχει τη συνάρτηση yylex η οποία υλοποιεί τον λεκτικό αναλυτή
- η συνάρτηση **αναγνωρίζει την επόμενη λεκτική μονάδα** και επιστρέφει ένα κωδικό που αντιστοιχεί σε αυτήν

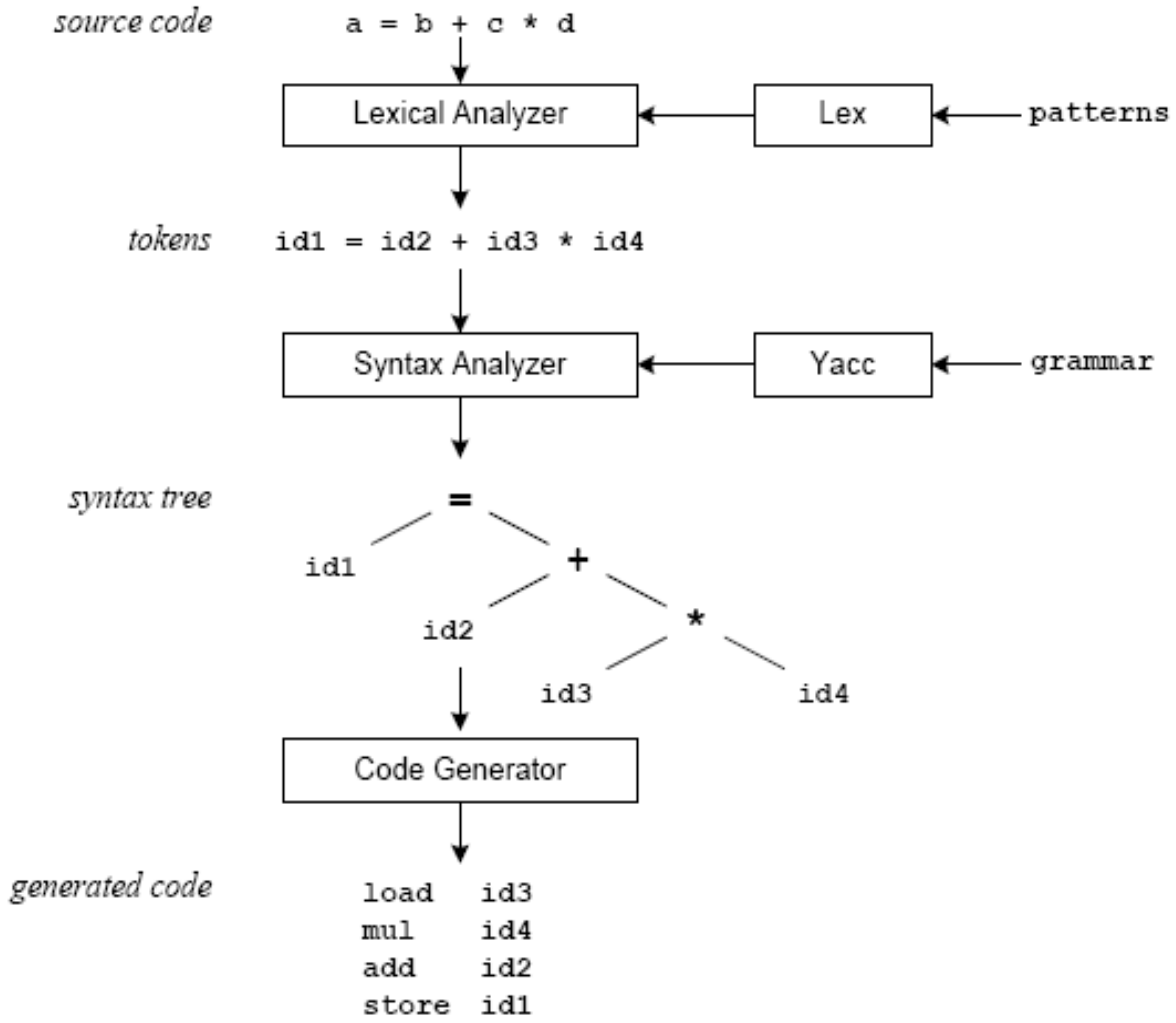
Lex

- αναγνωρίζει **κανονικές εκφράσεις**
- κάνει κάποια **ενέργεια** για κάθε κανονική έκφραση που αναγνωρίζει

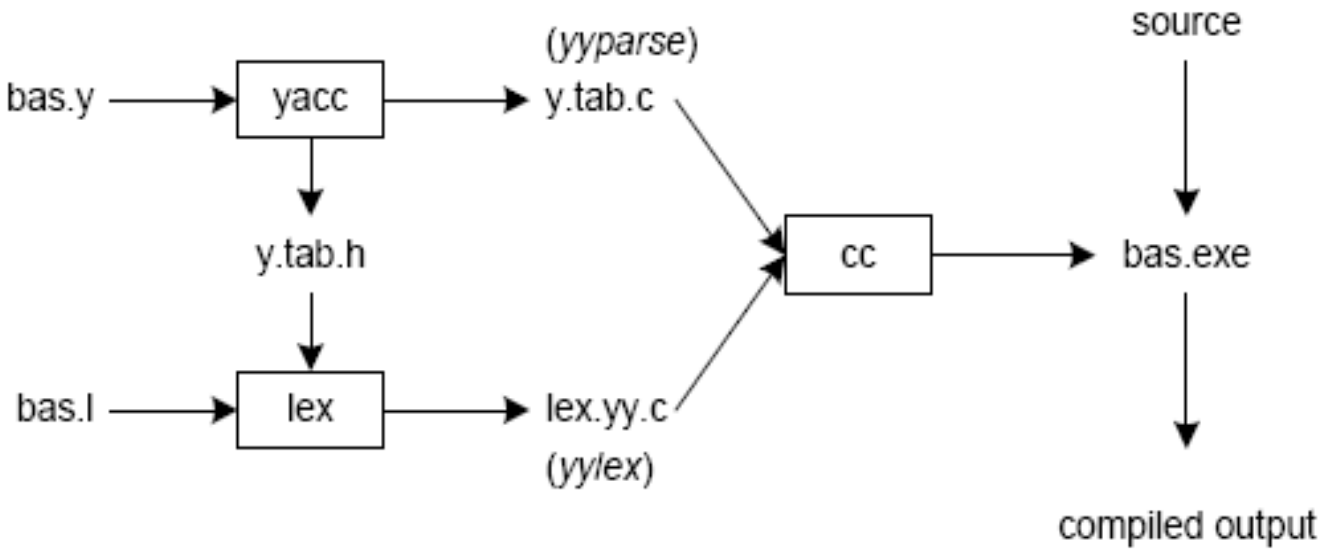
Σχηματικά



Μετάφραση με τα Εργαλεία Lex-Yacc

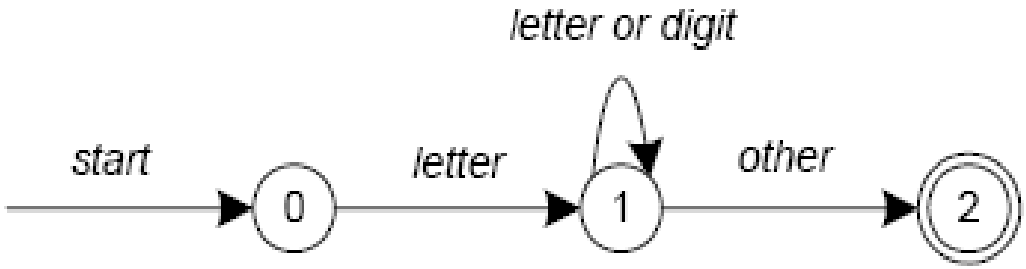


Μετάφραση με τα Εργαλεία Lex-Yacc




```
yacc -d bas.y                          # create y.tab.h, y.tab.c
lex bas.l                              # create lex.yy.c
cc lex.yy.c y.tab.c -obas.exe -ll -ly  # compile/link
```

Αναγνώριση Προτύπων – Κανονικές Εκφράσεις




```
letter(letter|digit)*
```

Κανονικές Εκφράσεις που Αντιστοιχούν σε Κώδικα

```
digit  [0-9]
letter [a-zA-Z]
%%
{letter}({letter}|{digit})*       printf("id: %s\n", yytext);
\n                                printf("new line\n");
%%
main() {
       yylex();
}
```

Από τον Lex στη C

- παράγεται το αρχείο lex.yy.c το οποίο
  - διαβάζει μία **συμβολοσειρά εισόδου**
  - χωρίζει την είσοδο σε **μικρότερες συμβολοσειρές**, αναγνωρίζοντας κανονικές εκφράσεις
  - αν χρειάζεται **μεταφέρει** την είσοδο στην έξοδο

Τελεστές

- “ \ [ ] ^ ? . - * + | ( ) $ / { } % < >
  - **δεσμευμένοι** χαρακτήρες
  - οποιοσδήποτε άλλος χαρακτήρας θεωρείται **κείμενο**
- οι παραπάνω τελεστές αν θέλουμε να χρησιμοποιηθούν σαν χαρακτήρες πρέπει να **προηγείται** από αυτούς ο χαρακτήρας «\» , δηλαδή **«\\» --> «\»**

Ομάδες χαρακτήρων

- [abc] **οποιοσδήποτε** από τους **χαρακτήρες** a b ή c
- [a-z] **οποιοσδήποτε** από τους **χαρακτήρες**
  a b c d e f g h i j k l m n o p q r s t u v w x y z
- [-+0-9] **προσημασμένος μονοψήφιος αριθμός**
- [^a-zA-Z] **οτιδήποτε** δεν είναι **γράμμα**

Τυχαίος Χαρακτήρας

- Ο χαρακτήρας **«.»** σημαίνει **οποιοσδήποτε** χαρακτήρας **εκτός new line**

Ομάδες Χαρακτήρων

- a? **καμία ή μία** εμφάνιση του a
- a* **καμία ή περισσότερες** εμφανίσεις του a
- a+ **μία ή περισσότερες** εμφανίσεις του a

- παραδείγματα
  - ab?c --> ac or abc
  - [a-z]+ --> μη κενές συμβολοσειρές από μικρά γράμματα
  - [a-zA-Z] [a-zA-Z0-9]* --> συμβολοσειρές από γράμματα και αριθμούς που ξεκινάνε από γράμμα

Προτεραιότητα Τελεστών

- από μεγαλύτερη σε μικρότερη
  - * ? +
  - παράθεση
  - |
- όλοι οι τελεστές συσχετίζονται προς τα αριστερά (left associative)
- παράδειγμα:
  - a*b | c*d --> ((a*)b) | (c*)d))

Σύνταξη Lex

- Μέρος Α: **Ορισμοί**
- Μερός Β: **Κανόνες**
- Μέρος Γ: **Συναρτήσεις**

Σύνταξη Lex

- Μέρος Α
  - **Σχόλια** με τη σύμβαση της C

    /* This is a comment */
  - **Μνημονικά** ονόματα
    - χρησιμοποιούνται στο Β μέρος ως συντομογραφίες για κανονικές εκφράσεις, π.χ.

      letter [A-Za-z]

      digit [0-9]
  - **Δηλώσεις** αρχικών καταστάσεων

Σύνταξη Lex

- **Κώδικάς C**
  - περικλείεται από %{ και %}
  - συνήθως περιέχει δηλώσεις μακροεντολών, τύπων δεδομένων και μεταβλητών που χρησιμοποιούνται από το λεκτικό αναλυτή

```
%{              #define MAX_LEXEME 256
                #define T_EOF 0
                          #define T_zero 1
                #define T_one

                typedef struct {
                   char lexeme[MAX_LEXEME];
                   int lineNumber, charPosition;
                } tokenInfo

                int currentLine=1, currentChar=1;
 %}
```

Σύνταξη Lex

- Μέρος Β
  - αποτελείται από κανόνες που περιγράφουν **ομάδες λεκτικών μονάδων**, ενώ σε κάθε κανόνα αντιστοιχίζονται και κάποιες ενέργειες

    Κανονική έκφραση 1 ενεργεια 1

    Κανονική έκφραση 2 ενεργεια 2

    ...

    Κανονική έκφραση N ενεργεια N
  - διαβάζονται χαρακτήρες από το αρχείο εισόδου έως ότου αναγνωριστεί το **μακρύτερο πρόθεμα** από μία από τις παραπάνω κανονικές εκφράσεις. Αν το πρόθεμα αυτό περιγράφεται από περισσότερες της μίας κανονικής έκφρασης, τότε επιλέγεται το πρώτο

Σύνταξη Lex

- Μέρος Β

Παράδειγμα

```
%%
        .       charcount++;
        \n      { charcount++; linecount++ }
      %%
```

Σύνταξη Lex

- Μέρος Γ
  - Κώδικας C

Παράδειγμα

```
int main()
{
   yylex();
   printf("There were %d characters in %d lines\n",
                                    charcount,linecount);
   return 0;
}
```

Ολοκληρωμένο Μικρό Παράδειγμα

```
%{
    int nchar, nword, nline;
%}
%%
\n          { nline++; nchar++; }
[^ \t\n]+   { nword++, nchar += yyleng; }
.           { nchar++; }
%%
int main(void) {
    yylex();
    printf("%d\t%d\t%d\n", nchar, nword, nline);
    return 0;
}
```

Σύμβολα Περιγραφής

x	matches the character x
.	(Period) matches any single character except a newline.
\n	matches a newline character
* or "*"	\ is used both as an escape character, so that you can use a reserved character as a literal, and to specify certain control characters, such as newline characters (\n) and tabs (\t). If the \ does not specify a control character, then it escapes the character. For example, * is a literal asterisk, rather than an asterisk meaning 0 or more occurrences of a regular expression. Alternatively you can use quotes (" ") to specify that a reserved character should be interpreted literally as that character.
$	By itself, $ is a special symbol meaning end of input (EOF). For example, `"$" { return 0; }`. Normally you do not care about EOF unless you need to do some sort of special processing, such as switching to another input file.

Σύμβολα Περιγραφής

r$	When placed at the end of a regular expression, $ specifies that the string that matches the regular expression r must be at the end of the current line of input.
[xyz]	a character class that matches any of the characters between the []'s. In this case the character class matches any of x, y, or z
[a-zA-Z]	the '-' denotes a range of ascii characters. This specification matches any lower or upper case letter. Do not make the mistake of writing [a-Z] because there are ascii characters between lowercase 'z' and uppercase 'A' that would be included in the pattern.
[0-9]	any single digit
[\t\n\r\f]	matches any whitespace character. \r and \f stand for "return" and "form feed" and are often present in Windows generated files.
[^A-Z]	A ^ that is the *first* character inside the character class negates that character class, or alternatively, says any character but the characters in that character class. In this case [^A-Z] says anything except an uppercase letter

Σύμβολα Περιγραφής

^r	When placed at the beginning of a pattern, the ^ says that the string which matches the regular expression r must start at the beginning of a line of input.
[a-z]{-}[aeiou]	The set difference operator (-) subtracts anything in the second character class from the first character class. In this case the pattern specifies the consonants.
r*	0 or more r's, where r is any regular expression.
r+	1 or more r's, where r is any regular expression.
r?	0 or 1 r's, where r is any regular expression. You may also think of ? as saying that the regular expression is optional. For example, **-?[0-9]** matches a single digit with an optional leading minus sign.
r{2,5}	Matches anywhere from 2 to 5 r's
r{4,}	Matches 4 or more r's
r{4}	Matches exactly 4 r's
rs	the concatenation of the regular expressions r and s. You can also think of the pattern as r followed by s.

Σύμβολα Περιγραφής

rs	the concatenation of the regular expressions r and s. You can also think of the pattern as r followed by s.
r | s	either r or s (i.e., the union operation).
[0-9]+	any number
. | \n	matches any character.
(brad|bvz)*	parentheses are used to group regular expressions and to override precedence. For example, `brad|bvz*` would typically match either "brad" or "bv" followed by 0 or more z's. To instead match 0 or more occurrences of either "brad" or "bvz", you would use parentheses: `(brad|bvz)*`.
{DIGIT}+"."{DIGIT}*	A name that is placed between curly braces ({}) will be replaced by its associated pattern from the definitions section. If DIGIT were defined as [0-9] in the definitions section, then this pattern specifies a number that consists of 1 or more digits, followed by a period, followed by 0 or more digits. Note that the decimal point had to be placed in quotes to prevent it from being interpreted as a pattern that matches any single character.
<s>r	A regular expression that is active only when state `s` is enabled. See Section States for more details.

Σύμβολα Περιγραφής

<*>r	A regular expression that is active in any state.
<s1,s2,s3>r	A regular expression that is active only when state s1, s2, or s3 is active.

Παραδείγματα Εκφράσεων

Expression	Matches
`abc`	`abc`
`abc*`	`ab abc abcc abccc ...`
`abc+`	`abc abcc abccc ...`
`a(bc)+`	`abc abcbc abcbcbc ...`
`a(bc)?`	`a abc`
`[abc]`	one of: `a`, `b`, `c`
`[a-z]`	any letter, a-z
`[a\-z]`	one of: `a`, `-`, `z`
`[-az]`	one of: `-`, `a`, `z`
`[A-Za-z0-9]+`	one or more alphanumeric characters
`[ \t\n]+`	whitespace
`[^ab]`	anything except: `a`, `b`
`[a^b]`	one of: `a`, `^`, `b`
`[a|b]`	one of: `a`, `|`, `b`
`a|b`	one of: `a`, `b`

Παράδειγμα Κώδικα – Αριθμοί Γραμμών

```
%{
    int yylineno;
%}
%%
^(.*)\n    printf("%4d\t%s", ++yylineno, yytext);
%%
int main(int argc, char *argv[]) {
    yyin = fopen(argv[1], "r");
    yylex();
    fclose(yyin);
}
```

Παράδειγμα Κώδικα – Μέτρηση Αναγνωριστικών

```
digit     [0-9]
letter    [A-Za-z]
%{
    int count;
%}
%%
    /* match identifier */
{letter}({letter}|{digit})*        count++;
%%
int main(void) {
    yylex();
    printf("number of identifiers = %d\n", count);
    return 0;
}
```

Παράδειγμα – Διόρθωση Περιττών Κενών

```
punct [,.;:!?]
text [a-zA-Z]

%%

")"" "+/{punct}          {printf(")");}
")"/{text}               {printf(") ");}
{text}+" "+/")"          {while (yytext[yyleng-1]==' ') yyleng--; ECHO;}

({punct}|{text}+)/"("    {ECHO; printf(" ");}
"("" "+/{text}           {while (yytext[yyleng-1]==' ') yyleng--; ECHO;}

{text}+" "+/{punct}      {while (yytext[yyleng-1]==' ') yyleng--; ECHO;}

^" "+                    ;
" "+                     {printf(" ");}
.                        {ECHO;}
\n/\n\n                  ;
\n                       {ECHO;}
```

Μικρό Παράδειγμα Λεκτικού Αναλυτή

```
%{
#include "y.tab.h"
%}

%%

","      { return COMMATK; }

program     { return PROGRAMTK; }
declare     { return DECLARETK; }
enddeclare  { return ENDDECLARETK; }
begin       { return BEGINTK; }
end         { return ENDTK; }

[a-z]+      {  strcpy(yylval.strV,yytext);
            return STRING;
          }
.           ;
"\n"        ;

%%
```

Λεκτικός Αναλυτής, όχι πολύ κομψός

```
#define idtk 1
#define numbertk 2
#define iftk 3
#define elsetk 4
#define whiletk 5
#define returntk 6
#define fortk 7
#define totk 8
#define switchtk 9
#define casetk 10
#define assigntk 11
#define equaltk 12
#define smallertk 13
#define smallerequaltk 14
#define largertk 15
#define largerequaltk 16
#define differenttk 17
#define plustk 18
#define minustk 19
#define timestk 20
#define overtk 21
#define rightpartk 22
#define leftpartk 23
#define rightbracktk 24
#define leftbracktk 25
#define rightarraytk 26
#define leftarraytk 27
#define semicolumntk 28
#define exittk 29
#define andtk 30
#define ortk 31
#define nottk 32
#define vartk 33
#define calltk 34
#define inttk 35
#define floattk 36
#define intk 37
#define outtk 38
#define inouttk 39
#define commatk 40
```

Λεκτικός Αναλυτής, όχι πολύ κομψός

letter [a-zA-Z]

digit [0-9]

delim [" "\t\n]

id {letter}({letter}|{digit})*

number {digit}+

Λεκτικός Αναλυτής, όχι πολύ κομψός

```
{delim} ;

"{" return(leftbracktk);
"}" return(rightbracktk);
"(" return(leftpartk);
")" return(rightpartk);
"[" return(leftarraytk);
"]" return(rightarraytk);

";" return(semicolumntk);
"," return(commatk);

else return(elsetk);
if return(iftk);
while return(whiletk);
return return(returntk);
for return(fortk);
to return(totk);
switch return(switchtk);
case return(casetk);
exit return(exittk);
call return(calltk);

"<=" return(smallerequaltk);
"<>" return(differenttk);
">=" return(largerequaltk);
":=" return(assigntk);
"<" return(smallertk);
">" return(largertk);
"=" return(equaltk);

"+" return(plustk);
"-" return(minustk);
"*" return(timestk);
"/" return(overtk);

and return(andtk);
or return(ortk);
not return(nottk);

int return(inttk);
float return(floattk);

var return(vartk);
inout return(inouttk);
in return(intk);
out return(outtk);

{id} return(idtk);
{number} return(numbertk);
```

Λεκτικός Αναλυτής μιας μικρής Pascal

```
/* scanner for a toy Pascal-like language */

%{
/* need this for the call to atof() below */
#include <math.h>
%}

DIGIT    [0-9]
ID       [a-z][a-z0-9]*
```

Λεκτικός Αναλυτής μιας μικρής Pascal

```
%%

{DIGIT}+    {
            printf( "An integer: %s (%d)\n", yytext,
                    atoi( yytext ) );
            }

{DIGIT}+"."{DIGIT}*         {
            printf( "A float: %s (%g)\n", yytext,
                    atof( yytext ) );
            }

if|then|begin|end|procedure|function        {
            printf( "A keyword: %s\n", yytext );
            }

{ID}        printf( "An identifier: %s\n", yytext );

"+"|"-"|"*"|"/"   printf( "An operator: %s\n", yytext );

"{"[^}\n]*"}"     /* eat up one-line comments */

[ \t\n]+          /* eat up whitespace */

.           printf( "Unrecognized character: %s\n", yytext );

%%
```

Λεκτικός Αναλυτής μιας μικρής Pascal

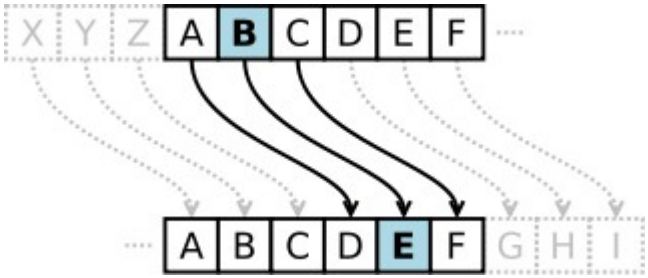
```
main( argc, argv )
int argc;
char **argv;
    {
    ++argv, --argc;  /* skip over program name */
    if ( argc > 0 )
            yyin = fopen( argv[0], "r" );
    else
            yyin = stdin;

    yylex();
    }
```

Κρυπτογραφία Καίσαρα

```
%%
[a-z]    { char ch = yytext[0];
           ch += 3;
           if (ch > 'z') ch -= ('z'+1-'a');
           printf ("%c", ch);
         }

[A-Z]    { char ch = yytext[0];
           ch += 3;
           if (ch > 'Z') ch -= ('Z'+1-'A');
           printf ("%c", ch);
         }
%%
```



Ρήμα ή όχι?

```
%{
/* this sample demonstrates very simple word recognition: verbs & other */
%}
%%

[\t ]+          /* ignore whitespace */ ;

is              |
am              |
are             |
were            |
was             |
be              |
being           |
been            |
do              |
does            |
did             |
will            |
would           |
should          |
can             |
could           |
has             |
have            |
had             |
go              { printf ("\"%s\" is a verb\n", yytext); }
[a-zA-Z]+       { printf ("\"%s\" is not a verb\n", yytext); }
.               |
\n              ECHO; /* which is the default anyway */
%%
int main (void) {
        return yylex ();
}
/* This is the "default main program" anyway, so it could be omitted. */
```

Τι κάνει?

```
%{
#include <strings.h>

int longest = 0;
char longword[60];
%}

%%
[a-zA-Z]+       { if (yyleng > longest) {
                      longest = yyleng;
                      strcpy (longword, yytext);
                      }
                }
.       |
\n      ;
%%
```

Βρίσκει τη Μεγαλύτερη Λέξη

```
%{
#include <strings.h>

int longest = 0;
char longword[60];
%}

%%
[a-zA-Z]+       { if (yyleng > longest) {
                      longest = yyleng;
                      strcpy (longword, yytext);
                      }
                }
.       |
\n      ;
%%
int main (void) {
   yylex ();
   printf ("The longest word was \"%s\", which was %d characters long.\n",
            longword, longest);
   return 0;
   }
```

Το ίδιο ...

```c
#include <stdio.h>
#include <strings.h>

#define MAXLEN 60

int main (void) {
   char word[MAXLEN], longword[MAXLEN];
   int wordlen;
   int longest = 0;
   int ch;

   ch = getchar ();
   while (ch != EOF) {
      /* First, skip anything other than letters. */
      while (! isalpha (ch) && ch != EOF) {
         ch = getchar ();
         }

      /* Now we've got the beginning of a word. */
      wordlen = 0;
      while (isalpha (ch)) {
         if (wordlen < MAXLEN-1) { /* don't want to overflow the buffer */
            word[wordlen++] = ch;
            }
         ch = getchar ();
         }
      word[wordlen] = '\0'; /* finish it with a null character */

      if (wordlen > longest) {
         longest = wordlen;
         strcpy (longword, word);
         }
      }

   /* Reached end of file.  Print the answer. */
   printf ("The longest word was \"%s\", which was %d characters long.\n",
           longword, longest);
   return 0;
   }
```

Romans

```
WS      [ \t]+

%%
        int total=0;

I       total += 1;
IV      total += 4;
V       total += 5;
IX      total += 9;
X       total += 10;
XL      total += 40;
L       total += 50;
XC      total += 90;
C       total += 100;
CD      total += 400;
D       total += 500;
CM      total += 900;
M       total += 1000;

{WS}    |
\n      return total;
%%
int main (void) {
   int first, second;

   first = yylex ();
   second = yylex ();

   printf ("%d + %d = %d\n", first, second, first+second);
   return 0;
   }
```

Πιστωτικές Κάρτες

```
%option noyywrap
%{
/* * * * * * * * * * * *
 * * * DEFINITIONS * * *
 * * * * * * * * * * * */
%}

%{
// recognize whether or not a credit card number is valid
int line_num = 1;
%}

digit [0-9]
group {digit}{4}
%%
```

Πιστωτικές Κάρτες

```
%{
/* * * * * * * * * *
 * * * RULES * * *
 * * * * * * * * * */
%}
   /* The carat (^) says that a credit card number must start at the
      beginning of a line and the $ says that the credit card number
      must end the line. */
^{group}([ -]?{group}){3}$  { printf(" credit card number: %s\n", yytext); }

   /* The .* accumulates all the characters on any line that does not
      match a valid credit card number */
.* { printf("%d: error: %s \n", line_num, yytext); }
\n { line_num++; }
%%

/* * * * * * * * * * *
 * * * USER CODE * * *
 * * * * * * * * * * *
 */
int main(int argc, char *argv[]) {
  yylex();
}
```

Τι κάνει?

```
%%
            int k;
[0-9]+      {
            k = atoi(yytext);
            if (k%7 == 0)
                  printf("%d", k+3);
            else
                  printf("%d",k);
            }
```

Τίποτα χρήσιμο

```
%%
            int k;
[0-9]+      {
            k = atoi(yytext);
            if (k%7 == 0)
                  printf("%d", k+3);
            else
                  printf("%d",k);
            }
```

αυξάνει κατά 3, ό,τι διαιρείται με το 7

Τι κάνει?

```
            int lengs[100];
%%
[a-z]+      lengs[yyleng]++;
.           |
\n          ;
%%
yywrap()
{
int i;
printf("Length  No. words\n");
for(i=0; i<100; i++)
     if (lengs[i] > 0)
          printf("%5d%10d\n",i,lengs[i]);
return(1);
}
```

Ιστόγραμμα

```
            int lengs[100];
%%
[a-z]+      lengs[yyleng]++;
.           |
\n          ;
%%
yywrap()
{
int i;
printf("Length  No. words\n");
for(i=0; i<100; i++)
     if (lengs[i] > 0)
          printf("%5d%10d\n",i,lengs[i]);
return(1);
}
```

Ευχαριστώ !!!