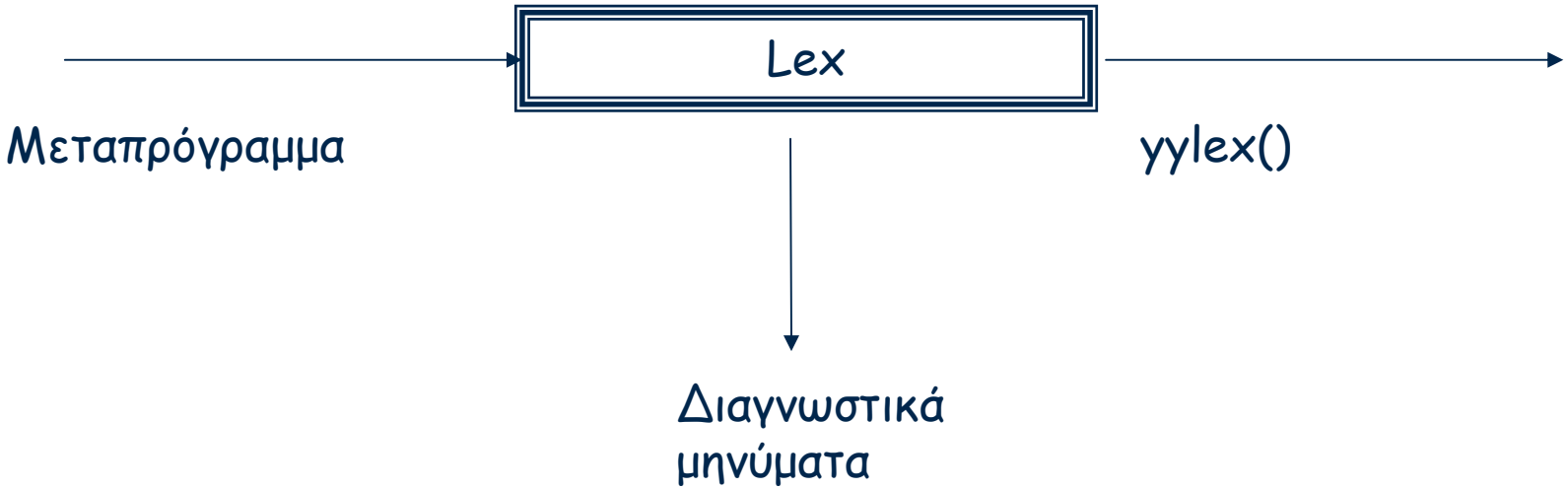
Λεκτική Ανάλυση
με το Εργαλείο LEX

Γιώργος Μανής

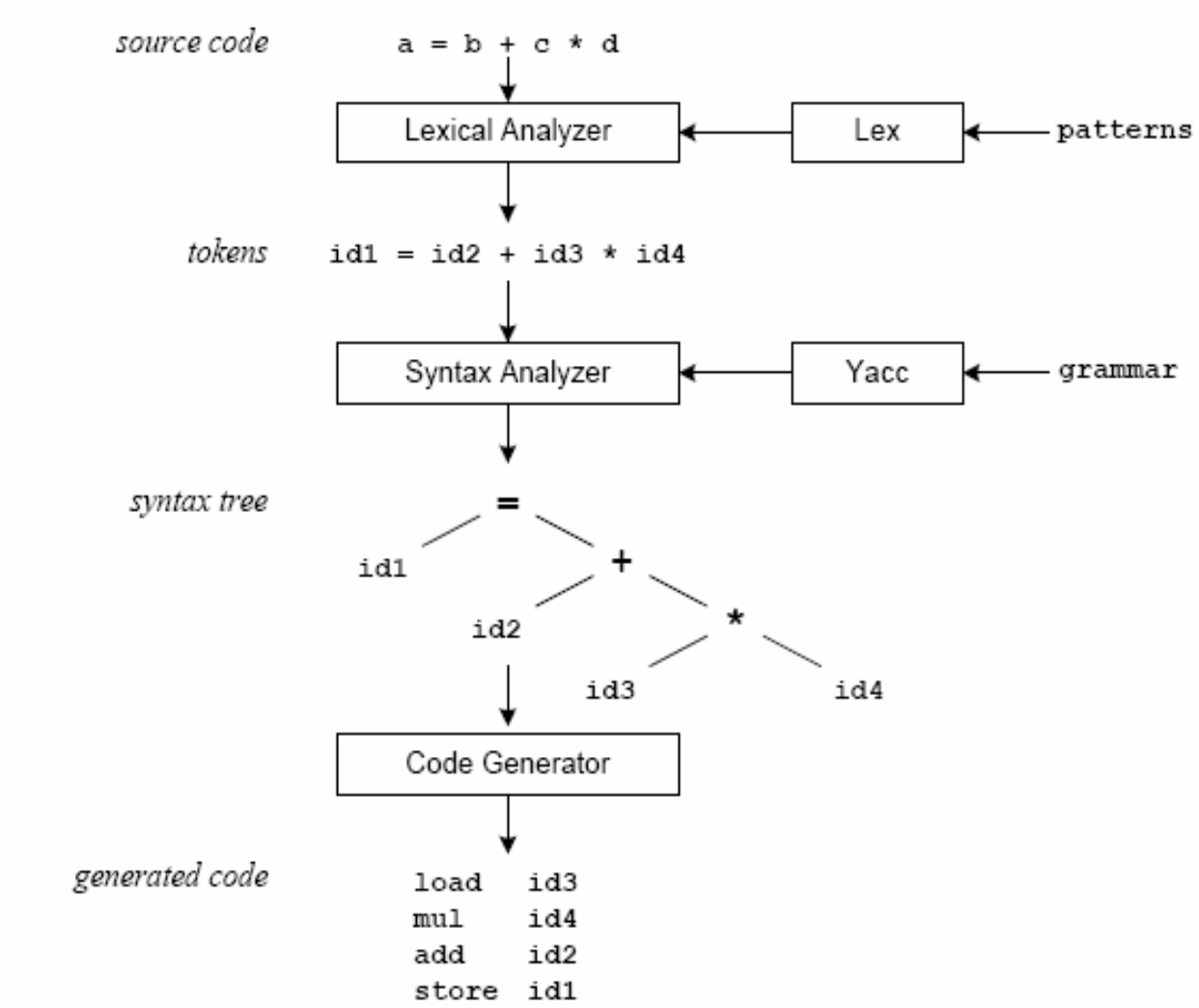
Γενικά

- 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

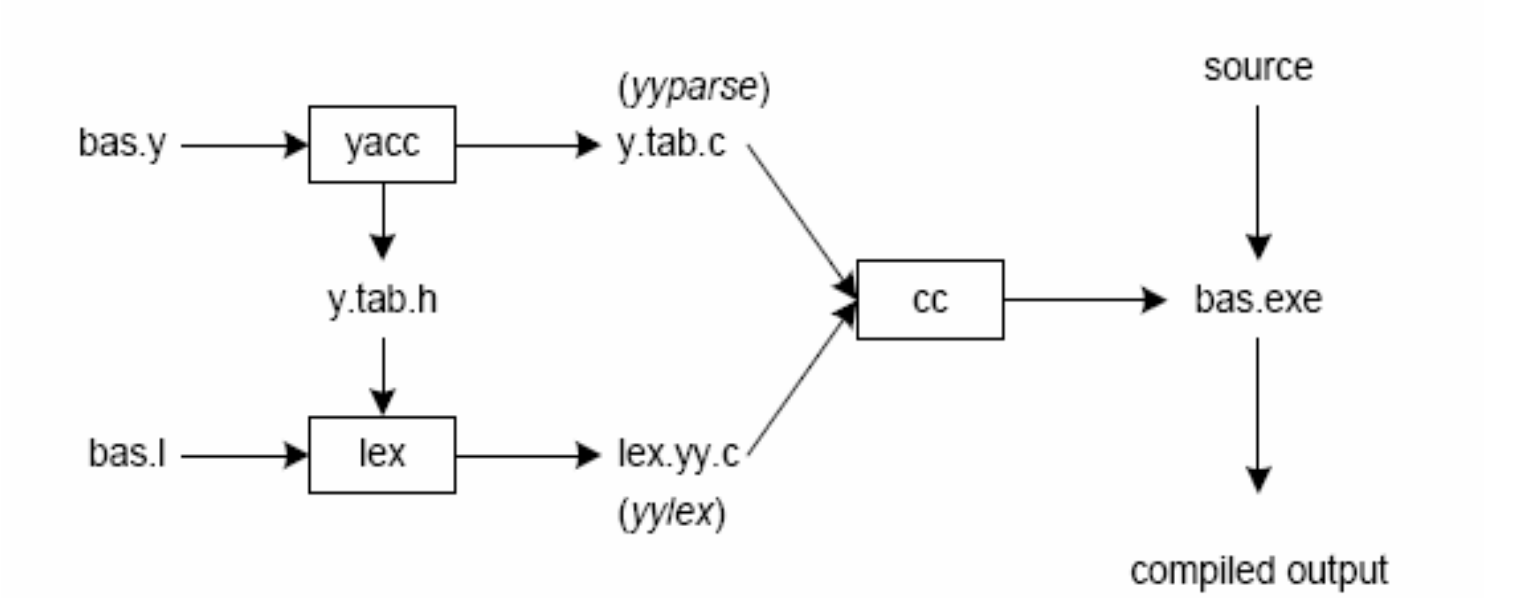
Σχηματικά



Μετάφραση με τα Εργαλεία Lex-Yacc

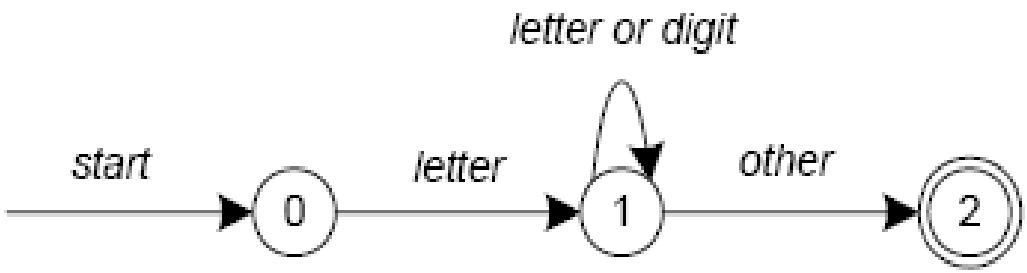


Μετάφραση με τα Εργαλεία Lex-Yacc




```
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    # compile/link
```

Αναγνώριση Προτύπων




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

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

Metacharacter	Matches
.	any character except newline
\n	newline
*	zero or more copies of the preceding expression
+	one or more copies of the preceding expression
?	zero or one copy of the preceding expression
^	beginning of line
$	end of line
a|b	a or b
(ab)+	one or more copies of ab (grouping)
"a+b"	literal "a+b" (C escapes still work)
[]	character class

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

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

Παράδειγμα

```
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;
}
```

Λεκτικός Αναλυτής

```
#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

letter [a-zA-Z]
digit [0-9]
delim [" "\t\n]

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

Λεκτικός Αναλυτής

```
{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);
```