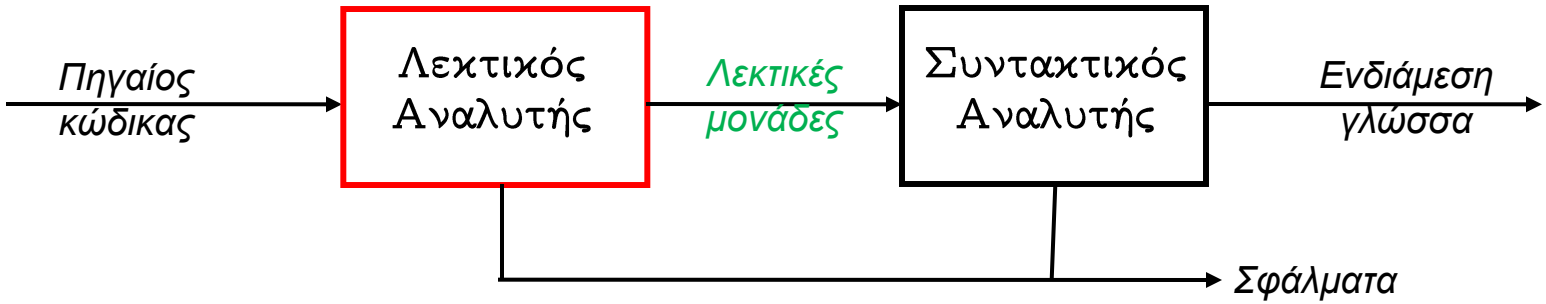
Αρχές Γλωσσών Προγραμματισμού & Μεταφραστών 2010-2011

Compiler Tools:

Lex / Flex

Φάση Λεκτικής Ανάλυσης (1)



Scanner (Λεκτικός Αναλυτής)

- *Είσοδος:* Το πηγαίο πρόγραμμα (μία συμβολοσειρά).
- *Έξοδος* : Λεκτικές μονάδες (tokens)
  - → **Token** : αναπαρίσταται από ένα ζεύγος <tokenName,attribute>
    - attribute : αριθμός, συμβολοσειρά, δείκτης στον πίνακα συμβόλων, δομή κ.λ.π . Το attribute είναι προαιρετικό.
    - tokenName : ένα αφηρημένο σύμβολο που εκφράζει μια κλάση χαρακτήρων και το οποίο αποτελεί είσοδο στον parser.
      - → π.χ. η ακολουθία χαρακτήρων x = x + y ; *γίνεται:*
        <id,x> <eq,=> <id,x> <plus_op,+> <id,y> <sc,; >

Φάση Λεκτικής Ανάλυσης (2)

Σχετικές Έννοιες:

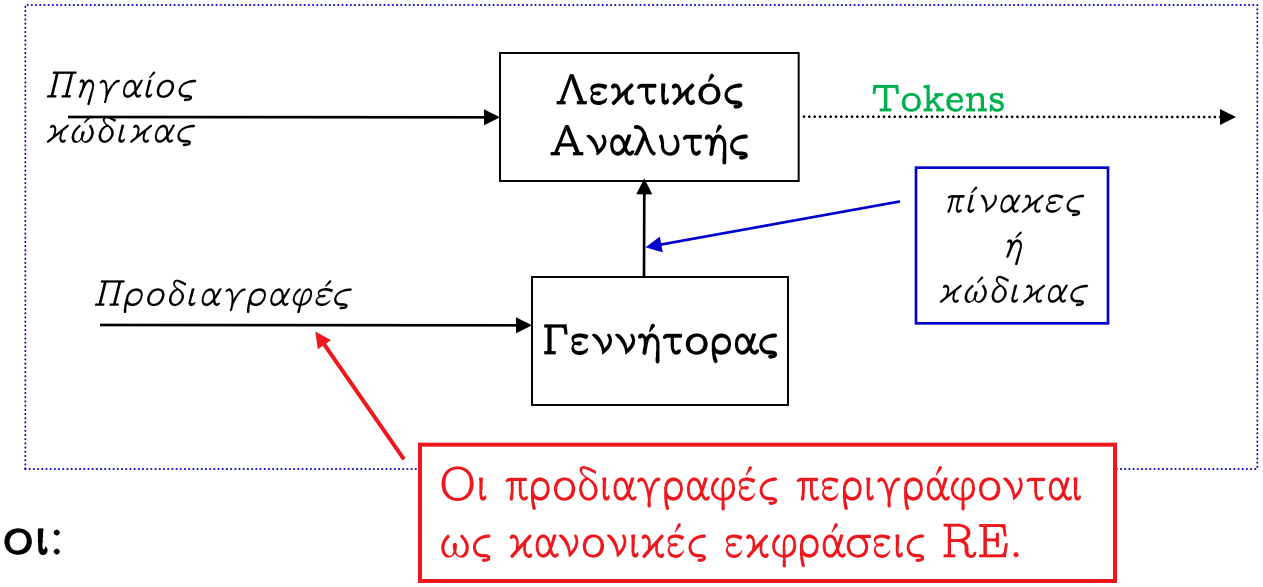
→Pattern: αποτελεί μια περιγραφή της μορφής που μπορούν να πάρουν τα λεξήματα (π.χ μία κανονική έκφραση)

→Lexeme (ή λέξημα) : μία ακολουθία χαρακτήρων που ταιριάζει με ένα pattern (π.χ. το keyword ***while***, ή το string *"Ceid")*

Λεκτική ανάλυση: Διαβάζει χαρακτήρες από την είσοδο και τους ταιριάζει με **patterns**, παράγοντας **lexemes** , τα οποία αντιστοιχίζονται σε **tokens** που αποτελούν την είσοδο του συντακτικού αναλυτή.

Γιατί είναι χρήσιμη η μελέτη Λεκτικών Αναλυτών?

- Θέλουμε να αποφύγουμε την κοπιαστική δουλειά της χειρονακτικής γραφής λεκτικών αναλυτών.
- Σε πολλές εφαρμογές όπως, grep, website filtering, διάφορες εντολές "find" κ.λ.π χρησιμοποιούνται **πεπερασμένα αυτόματα.**

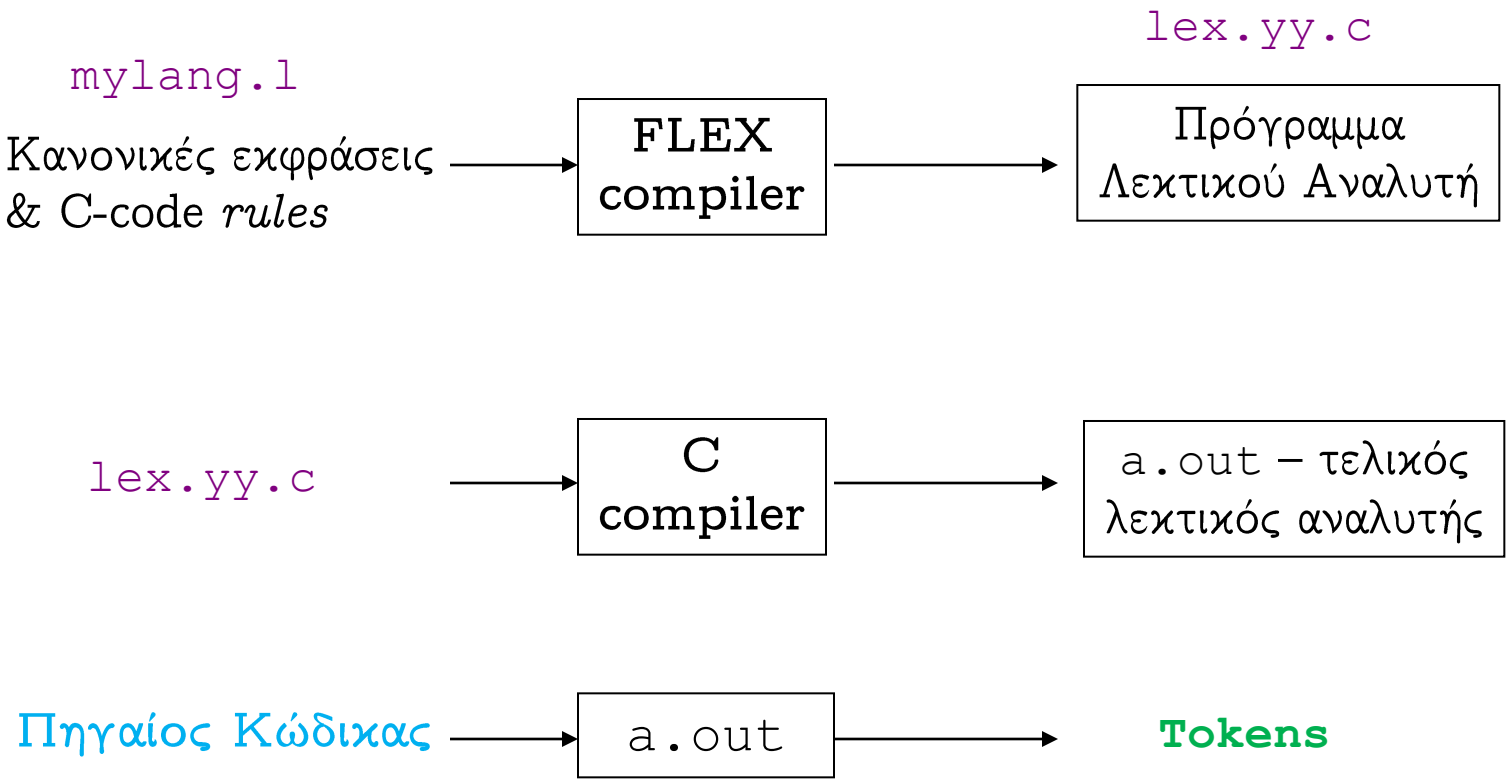


Στόχοι:

→ Απλοποίηση περιγραφής προδιαγραφών

→ Απλοποίηση της διαδικασίας υλοποίησης λεκτικών αναλυτών

Flex – Fast Lexical Analyzer



Κανονικές Εκφράσεις στον FLEX (1/2)

- Ο Flex υποστηρίζει μία επέκταση των Κ.Ε:

Κανονική Έκφραση	Ερμηνεία
x	ο χαρακτήρας x
.	Οποιοσδήποτε χαρακτήρας εκτός του χαρακτήρα αλλαγής γραμμής
“abc”	η ακολουθία χαρακτήρων abc
[abc]	Οποιοσδήποτε από τους χαρακτήρες a,b,c
[a-z]	Οποιοσδήποτε από τους χαρακτήρες a έως z
[^a-z]	Οποιοσδήποτε χαρακτήρας εκτός των a έως z
r*	Καμία ή περισσότερες επαναλήψεις της Κ.Ε r
	Μία ή περισσότερες επαναλήψεις της Κ.Ε r
r?	Μία ή καμία επανάληψη της Κ.Ε r

Κανονικές Εκφράσεις στον FLEX (2/2)

Κανονική Έκφραση	Ερμηνεία
r{i,j}	Από i έως j (0 < i < j) επαναλήψεις της r
rs	Ακολουθίες που προκύπτουν από παράθεση των r,s
(r)	Οι παρενθέσεις ορίζουν την εφαρμογή των τελεστών
r|s	Ακολουθίες που ικανοποιούν την r ή την s
^r	Ικανοποιείται μόνο αν η r βρίσκεται στην αρχή της γραμμής
r$	Ικανοποιείται μόνο αν η r βρίσκεται στο τέλος της γραμμής
<<EOF>>	Ικανοποιείται όταν συναντηθεί το EOF
Για την χρήση των ειδικών χαρακτήρων $,|,^,(,), κτλ ως κυριολεκτικών πρέπει να προηγηθεί backslash \	

Flex input file

- Δομή αρχείου εισόδου

```
declarations
%%
translation rules
%%
Auxiliary functions
```

Τμήμα 1 : **declarations** (ορισμοί)

- Το declarations section αποτελείται από:
  - **Ορισμούς ονομάτων** που χρησιμοποιούνται ως συντομογραφίες Κ.Ε και έχουν την μορφή:

    name regular_expression

    πχ:

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

  - **Κώδικα** που θέλουμε να συμπεριληφθεί στον παραγόμενο λεκτικό αναλυτή, συνήθως δηλώσεις μακροεντολών, μεταβλητών και τύπων δεδομένων μέσα σε '%{ %}' πχ:

```
%{
#include<stdio.h>
#define TK_IF 0
typedef struct{
char* lexeme;
int lineNumber, charPosition;
} tokenInfo;
%}
```

Αυτές οι γραμμές κώδικα αντιγράφονται όπως έχουν στην έξοδο

Τμήμα 2 : **rules** (κανόνες)

- Το τμήμα κανόνων είναι το κύριο τμήμα του προγράμματος και αποτελείται από κανόνες της μορφής:

```
pattern {action}
```

pattern ενέργεια

Το λέξημα που ταίριαξε στο pattern (char*)

πχ:

```
%%
{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 ); }
%%
```

- Όταν ικανοποιείται ένα pattern ο κανόνας ενεργοποιείται και ο κώδικας του action εκτελείται
- Όταν ικανοποιούνται περισσότεροι κανόνες ταυτόχρονα, επιλέγεται αυτός που καταναλώνει περισσότερους χαρακτήρες
- Εάν καταναλώνουν τον ίδιο αριθμό, επιλέγεται αυτός που έχει δηλωθεί πρώτος

Τμήμα 3 : **Auxiliary functions**

- Κώδικας ορισμένος από το χρήστη. Εδώ κρατάμε και οποιεσδήποτε συναρτήσεις έχουν ορισθεί από τον χρήστη και χρησιμοποιούνται στα actions του τμήματος 2. πχ:

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

lex input file

Η lexer function που παράγεται από το flex

Καταστάσεις στον Flex

- Ο Flex υποστηρίζει την υπό συνθήκη ενεργοποίηση ενός κανόνα μέσω της έννοιας της **κατάστασης.** Αρχικά ο Flex βρίσκεται στην κατάσταση *INITIAL.*
- Οι καταστάσεις δηλώνονται στο τμήμα ορισμών (π.χ `%s A_STATE`)
- Ένας υπό συνθήκη ενεργός κανόνας γράφεται ως εξής
  - `<A_STATE>regular_expression  action`
    ή ως:
  - `<A_STATE,ANOTHER_STATE> regular_expression  action`
    όπου ο κανόνας είναι ενεργός σε οποιαδήποτε από τις δύο καταστάσεις.
- Η μετάβαση σε μία κατάσταση γίνεται γράφοντας στο τμήμα action : `BEGIN(A_STATE);`

Περιβάλλον Flex

- Μερικές από τις κυριότερες διαθέσιμες μεταβλητές/συναρτήσεις:
  - `char *yytext` : περιέχει το κομμάτι του κειμένου που έχει ικανοποιήσει την κανονική έκφραση (λέξημα)
  - `int yyleng` : ένας ακέραιος που δηλώνει το μέγεθος του `yytext`
  - `FILE *yyin` : το προκαθορισμένο αρχείο εισόδου
  - `int yylex()`: η παραγόμενη συνάρτηση λεκτικής ανάλυσης· επιστρέφει το αναγνωριστικό της λεκτικής μονάδας που διαβάζει
  - Περισσότερα στο Manual του Flex...

Παράδειγμα (1/3)

- Θέλουμε να φτιάξουμε ένα απλό λεκτικό αναλυτή που αναγνωρίζει email διευθύνσεις.
- Ο αναλυτής θα πρέπει να βγάζει κατάλληλο μήνυμα λάθους αν εισαχθεί μη έγκυρο email.

Παράδειγμα (2/3)

```
%{
#include <stdio.h>
#define TK_EMAIL              257
#define TK_ILLEGAL_SEQ        258
int emailcount=0;
%}

letter    [a-zA-Z]
word      {letter}{letter}+
digit     [0-9]
other_chars         [.\-_]
id        {letter}({letter}|{digit}|{other_chars})*
domain    {word}\.{word}(\.{word})*
email     {id}\@{domain}
ws        [ \t\n]

%%

{email}   {  printf("I have found %d emails, and counting...\n",++emailcount); return
TK_EMAIL;}
{ws}      { /* ignore whitespaces */ }
.         { return TK_ILLEGAL_SEQ; }

%%
```

Παράδειγμα (2/2)

```
int main(int argc, char* argv) {
        int token;

        while ((token = yylex()) != 0) {
                if (token == TK_ILLEGAL_SEQ) {
                        printf("Encountered illegal sequence \n ");
                } else {
                        printf("Email: %s\n", yytext);
                }
        return 0;
}
```

- ΑΣΚΗΣΗ : Να τροποποιήσετε τον παραπάνω κώδικα ώστε να χρησιμοποιεί ως είσοδο ένα αρχείο με το όνομα `email_input` και να παράγει ως έξοδο ένα αρχείο `email_output` που περιλαμβάνει όλα τα emails του αρχείου εισόδου.

Ευχαριστώ πολύ