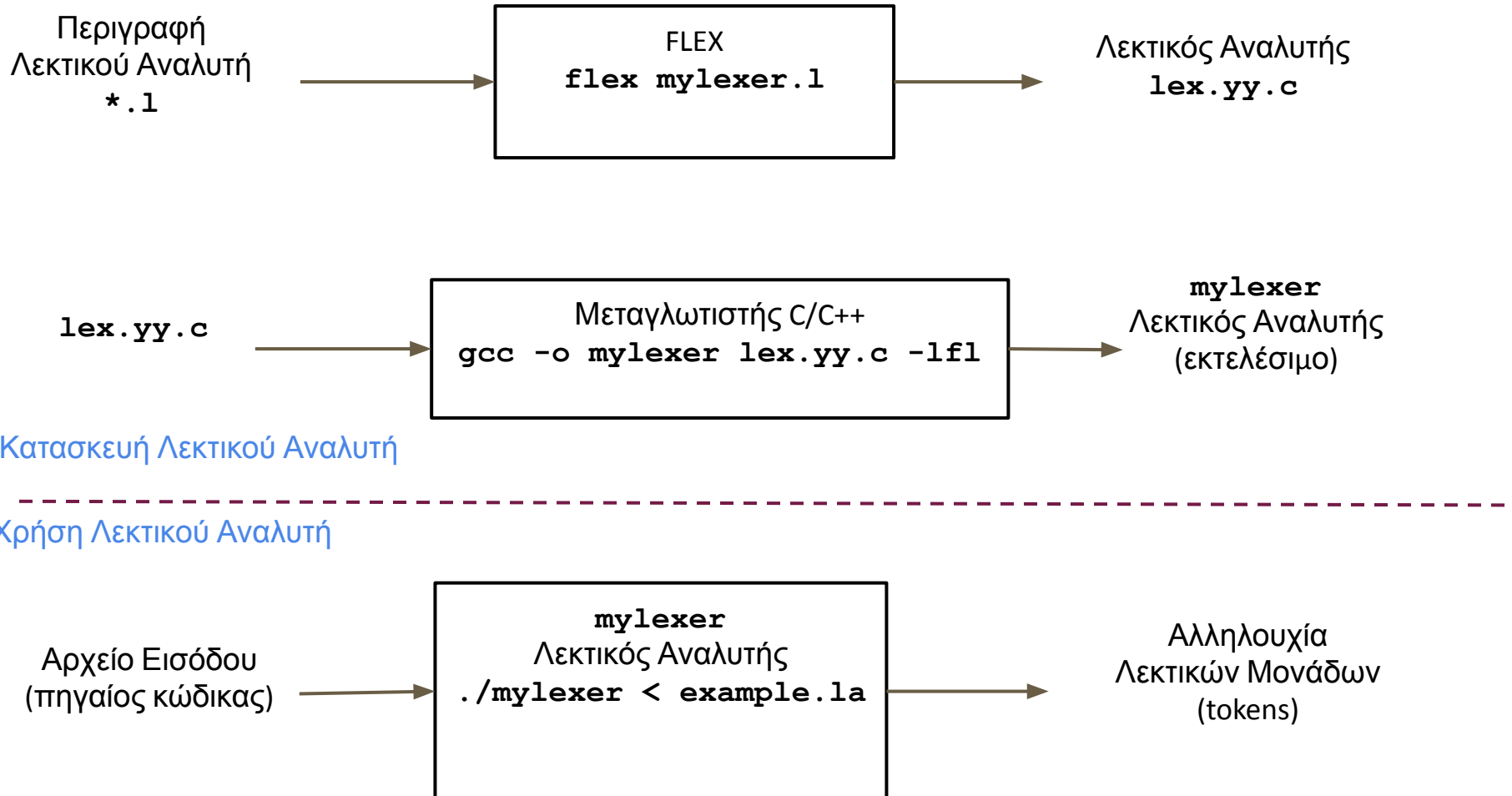

Θεωρία Υπολογισμού

Το Εργαλείο FLEX

Το Εργαλείο FLEX

- **FLEX: Fast LEXical analyzer generator**
- Εργαλείο για την αυτοματοποιημένη **παραγωγή** Σαρωτών - Λεκτικών Αναλυτών
- Σαρωτής (Scanner) - Λεκτικός Αναλυτής (Lexical Analyzer)
 - λογισμικό που εκτελεί λεκτική ανάλυση (lexical analysis) κειμένου (προγράμματος)
- Δέχεται ως είσοδο ένα αρχείο κειμένου (συνήθως, πηγαίος κώδικας) ως μια ακολουθία από χαρακτήρες και αναγνωρίζει **Λεκτικές Μονάδες (tokens)**
- Τρέχουσα έκδοση 2.6.4 (<https://github.com/westes/flex>)
- Ευκολία χρήσης, υψηλή απόδοση, ευελιξία και εκφραστικότητα
- Συνεργασία με Γεννήτριες Συντακτικών Αναλυτών (π.χ. bison)

Λειτουργία FLEX



Μορφή Αρχείου Εισόδου flex

Κάθε αρχείο flex αποτελείται από τρεις ενότητες χωρισμένες μεταξύ τους από μία γραμμή η οποία περιέχει μόνο το σύμβολο `%%` στην αρχή της:

definitions

`%%`

rules

`%%`

user code

Ενότητα Ορισμών (definitions)

- Περιέχει δηλώσεις
 - **ονοματισμένων ορισμών** (name definitions) και
 - **αρχικές συνθήκες** (start conditions)
- Οι ονοματισμένοι ορισμοί έχουν τη μορφή:
 - **name definition**
 - **name** : είναι μια λέξη η οποία ξεκινά με ένα γράμμα ή το χαρακτήρα κάτω παύλας (`_`) ακολουθούμενο από άλλα γράμματα, ψηφία, κάτω παύλες (`_`) ή παύλες (`-`)
 - **definition** : κανονική έκφραση που εκτείνεται μέχρι το τέλος της γραμμής
 - Ακολουθώς χρησιμοποιώντας το **name** μέσα σε αγκύλες **{name}** μπορούμε να αναφερόμαστε στο συγκεκριμένο ορισμό

Ενότητα Ορισμών (2/6)

- Παραδείγματα Ορισμών

`DIGIT` `[0-9]`

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

`{DIGIT}+"."` `{DIGIT}*`

- Ίδιο με την έκφραση:

`([0-9])+"."` `([0-9])*`

Ενότητα Ορισμών (3/6)

- Σχόλια
 - Ο flex υποστηρίζει σχόλια παρόμοια με τη γλώσσα C (C-style comments). Ο,τιδήποτε μεταξύ `/*` και `*/` θεωρείται σχόλιο. Ο flex μεταφέρει τα σχόλια αυτολεξεί στο παραγόμενο αρχείο κώδικα (`lex.yy.c`)
- Σύμβολα `%{` και `%}`
 - Ο,τιδήποτε περικλείεται μεταξύ των συμβόλων `%{` και `%}` επίσης αντιγράφεται αυτολεξεί στο παραγόμενο αρχείο κώδικα (`lex.yy.c`) χωρίς τα σύμβολα `%{` και `%}`
- Τα σύμβολα `%{` και `%}` πρέπει να εμφανίζονται μόνα τους σε ξεχωριστές γραμμές χωρίς ένθεση (unindented)

Ενότητα Ορισμών (4/6)

- Παράδειγμα %{ και %}

```
%{  
    #define TK_ID  0  
    #include <stdio.h>  
    int lineCount = 0;  
%}
```


Ενότητα Ορισμών (5/6)

- `%top` μπλοκ
 - Ένα `%top` μπλοκ είναι παρόμοιο με ένα `%{ ... %}` μπλοκ με τη διαφορά ότι τα περιεχόμενα του `%top` μπλοκ μεταφέρονται στην αρχή του παραγόμενου αρχείου κώδικα πριν από οποιονδήποτε ονοματισμένο ορισμό.
 - Οι χαρακτήρες `{` και `}` χρησιμοποιούνται για την οριοθέτηση του `%top` μπλοκ.
 - Το `%top` μπλοκ χρησιμεύει για τον ορισμών οδηγιών προς τον προεπεξεργαστή (preprocessor) ή για τη συμπερίληψη αρχείων πριν από τον παραγόμενο κώδικα.

Ενότητα Ορισμών (6/6)

- Παράδειγμα `%top` μπλόκ

```
%top{
```

```
/*This code goes at the "top" of the generated file*/
```

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

```
#include <string.h>
```

```
}
```

Ενότητα Κανόνων (rules)

- Η ενότητα **κανόνες** περιέχει μια σειρά κανόνων της μορφής:
 - **pattern action**
- **πρότυπο (pattern)**: κάποιος ονοματισμένος ορισμός (definition) ή κάποια κανονική έκφραση
- **ενέργεια (action)**: τοπικός κώδικας C/C++ ανάμεσα σε { και }
- Όταν η είσοδος ταιριάζει με κάποιο **pattern**, εκτελείται το **action**
- Σειρά εκτέλεσης κανόνων (προτεραιότητα): από πάνω προς τα κάτω

Κανόνες - Patterns (1/5)

Έκφραση Ταιριάζει με (matches)

x το χαρακτήρα **x**

\x αν **x** είναι ένας από τους χαρακτήρες **a, b, f, n, r, t, v**, την ANSI-C ερμηνεία του, αλλιώς τον ίδιο τον χαρακτήρα **x** (χρησιμοποιείται για τους ειδικούς χαρακτήρες - escape characters, όπως για παράδειγμα το '*')

. οποιονδήποτε χαρακτήρα εκτός του newline

\x2a το χαρακτήρα με δεκαεξαδική τιμή 2a

\123 το χαρακτήρα με οκταδική τιμή 123

Κανόνες - Patterns (2/5)

Έκφραση **Ταιριάζει με (matches)**

[xyz] έναν από τους χαρακτήρες **x**, **y**, **z** (κλάση χαρακτήρων)

[abj-oZ] ένα **a**, ή ένα **b**, ή οποιονδήποτε χαρακτήρα από το **j** έως και το **o** ή το **Z** (κλάση χαρακτήρων με ένα εύρος (range))

[^A-Z\n] οποιονδήποτε χαρακτήρα εκτός αυτών της κλάσης, σε αυτή την περίπτωση, οποιονδήποτε χαρακτήρα εκτός από τα κεφαλαία γράμματα ή το χαρακτήρα νέας γραμμής (newline)

r* καμιά ή περισσότερες εμφανίσεις της κανονικής έκφρασης **r**

r+ μια ή περισσότερες εμφανίσεις της κανονικής έκφρασης **r**

Κανόνες - Patterns (3/5)

Έκφραση	Ταιριάζει με (matches)
$r?$	καμιά ή μία εμφάνιση της κανονικής έκφρασης r (προαιρετικά r)
$r\{2, 5\}$	2 μέχρι 5 εμφανίσεις της κανονικής έκφρασης r
$r\{2, \}$	2 ή περισσότερες εμφανίσεις της κανονικής έκφρασης r
$\{name\}$	την επέκταση του ονοματισμένου ορισμού name , όπως αυτός ορίζεται στην ενότητα των ορισμών (definitions)
$"[xyz] \setminus "foo"$	τη συμβολοσειρά $[xyz] "foo"$

Κανόνες - Patterns (4/5)

Έκφραση **Ταιριάζει με (matches)**

$r s$	την κανονική έκφραση r ή την s (διάζευξη)
rs	την κανονική έκφραση r ακολουθούμενη από την s (παράθεση)
r/s	την κανονική έκφραση r αλλά μόνο όταν ακολουθείται από την κανονική έκφραση s (η οποία δε διαβάζεται στη yytext)
r	την κανονική έκφραση r αλλά μόνο στην αρχή μιας γραμμής
$r\$$	την κανονική έκφραση r αλλά μόνο στο τέλος μιας γραμμής

Κανόνες - Patterns (5/5)

Έκφραση	Ταιριάζει με (matches)
$\langle s \rangle r$	την κανονική έκφραση r αλλά μόνο με αρχική συνθήκη s
$\langle s1, s2, s3 \rangle r$	την κανονική έκφραση r αλλά μόνο με αρχική συνθήκη $s1$ ή $s2$ ή $s3$
$\langle * \rangle r$	την κανονική έκφραση r με οποιαδήποτε αρχική συνθήκη
$\langle \langle EOF \rangle \rangle$	το τέλος του αρχείου
$\langle s1, s2 \rangle \langle \langle EOF \rangle \rangle$	το τέλος του αρχείου αλλά μόνο με αρχική συνθήκη $s1$ ή $s2$

Ταίριασμα της Εισόδου (1/3)

- Όταν εκτελείται (run) ο Λεκτικός Αναλυτής, αναλύει την είσοδο για συμβολοσειρές (strings) οι οποίες ταιριάζουν με κάποια από τα δηλωθέντα patterns του
- Αν ταιριάζουν περισσότερα από ένα patterns επιλέγεται αυτό που ταιριάζει τους περισσότερους χαρακτήρες εισόδου (το μεγαλύτερο μήκος εισόδου)
- Αν ταιριάζουν περισσότερα από ένα patterns με το ίδιο μήκος εισόδου τότε επιλέγεται αυτό που έχει δηλωθεί πρώτο στο αρχείο flex

Ταίριασμα της Εισόδου (2/3)

- Όταν καθοριστεί το ταίριασμα (matching), το τμήμα της εισόδου που αντιστοιχεί στο ταίριασμα, ονομάζεται **λεκτική μονάδα (token)**, είναι διαθέσιμο μέσω της καθολικής μεταβλητής `char* yytext`
 - Το μήκος της λεκτικής μονάδας, δηλαδή το πλήθος των χαρακτήρων της, είναι διαθέσιμο μέσω της καθολικής μεταβλητής `int yyleng`
- Εκτελείται η ενέργεια του pattern που ταίριαξε με τμήμα της εισόδου
- Συνεχίζεται το σάρωμα (scanning) της εναπομείνουσας εισόδου για το επόμενο ταίριασμα

Ταίριασμα της Εισόδου (3/3)

- Αν δεν βρεθεί ταίριασμα, τότε εκτελείται ο *προκαθορισμένος κανόνας (default rule)*:
 - Ο επόμενος χαρακτήρας της εισόδου δίνεται στην έξοδο (standard output)
- Παρατήρηση
 - Μόνο του το σύμβολο `%%` σε ένα αρχείο flex δημιουργεί έναν πολύ απλό Λεκτικό Αναλυτή ο οποίος απλά αντιγράφει στην έξοδο την είσοδό του, χαρακτήρα προς χαρακτήρα

Κανόνες - Actions (1/2)

- Σε ένα κανόνα, κάθε **pattern** έχει μια αντίστοιχη **ενέργεια (action)** η οποία μπορεί να είναι οποιαδήποτε εντολή (statement) της C
- Αν η ενέργεια περιλαμβάνει πολλές εντολές της C οι οποίες εκτείνονται σε πολλές γραμμές, περικλείονται σε άγκιστρα
 - **pattern** {
 ...
}
- Η ενέργεια εκτελείται κάθε φορά που η είσοδος ταιριάζει (match) με το αντίστοιχο pattern
- Αν η ενέργεια είναι κενή, τότε η είσοδος που ταιριάζει, απλώς αγνοείται

Κανόνες - Actions (2/2)

- Ειδικές Οδηγίες (special directives)
 - **ECHO**
 - Εκτυπώνει στην έξοδο το τμήμα της εισόδου που αντιστοιχεί στο ταίριασμα, δηλαδή το περιεχόμενο της **yytext**
 - **BEGIN**
 - Ενεργοποιεί την αντίστοιχη αρχική συνθήκη (θα το δούμε στη συνέχεια)
 - **REJECT**
 - Απορρίπτει το ταίριασμα που έχει γίνει και προχωράει στο αμέσως επόμενο δυνατό ταίριασμα

Ενότητα Κώδικα (user code)

- Προαιρετική ενότητα
- Περιέχει βοηθητικό κώδικα C/C++
- Ενσωματώνεται ως έχει
- Ορισμός βοηθητικών συναρτήσεων
- Καλούνται από τις ενέργειες των κανόνων
- Περιέχει την `main` για αυτόνομο Λεκτικό Αναλυτή

Τυπική Μορφή Αρχείου Εισόδου

Συνήθως ένα αρχείο flex αποτελείται από τις τρεις προαναφερθείσες ενότητες χωρισμένες μεταξύ τους από μία γραμμή η οποία περιέχει μόνο το σύμβολο `%%` καθώς και από ένα μπλοκ `%{` και `%}` στην αρχή.

```
%{  
    C inclusion  
%}  
definitions  
%%  
rules  
%%  
user code
```

Παράδειγμα (lexer-1.1)

```
%{  
    #include <stdio.h>  
}%  
  
NUMBER    [+|-]? ([0-9]+\.[0-9]*| [0-9]*\.[0-9]+)  
ID         [a-zA-Z_][0-9a-zA-Z_]*  
  
%%  
  
{NUMBER}    { printf("Found the number %s\n", yytext); }  
{ID}        { printf("Found the identifier %s\n", yytext); }  
.  
{.}        { printf("Error\n"); }  
  
%%  
  
int main() {  
    yylex();  
}
```


Δημιουργία Λεκτικού Αναλυτή

- Παραγωγή κώδικα
 - `flex lexer-1.1`
- Μεταγλώττιση
 - `gcc lex.yy.c -lfl`
- Εκτέλεση
 - `./a.out < example-1_test-1.in`

Αρχικές Συνθήκες (Start Conditions)

- Μηχανισμός για την υπό συνθήκη ενεργοποίηση κανόνων
 - Κάθε κανόνας του οποίου το pattern έχει πρόθεμα `<sc>` θα ενεργοποιηθεί μόνο εφόσον ο Λεκτικός Αναλυτής βρίσκεται στην αρχική συνθήκη με όνομα `sc`
 - Για παράδειγμα ο κανόνας

```
<STRING>[^"]*      { /* eat up the string body ... */  
                      ...  
                      }
```

θα ενεργοποιηθεί μόνο εφόσον ο Λεκτικός Αναλυτής βρίσκεται στην αρχική συνθήκη `STRING`

Αρχικές Συνθήκες (2)

- Οι αρχικές συνθήκες δηλώνονται στην ενότητα των ορισμών χρησιμοποιώντας τις οδηγίες `%s` ή `%x` ακολουθούμενες από μια λίστα ονομάτων

```
/* inclusive start condition */
```

- `%s` **CONDITION**

```
/* exclusive start condition */
```

- `%x` **CONDITION**

Αρχικές Συνθήκες (3)

- *εγκλειστική (inclusive) %s CONDITION*
 - Ενεργοί οι κανόνες με συνθήκη **CONDITION** ή χωρίς συνθήκη
- *Αποκλειστική (exclusive) %x CONDITION*
 - Ενεργοί **μόνο** οι κανόνες με συνθήκη **CONDITION**
- Οι αποκλειστικές αρχικές συνθήκες επιτρέπουν τον καθορισμό “mini-scanners” οι οποίοι σαρώνουν τμήματα της εισόδου τα οποία είναι συντακτικά διαφορετικά από την υπόλοιπη είσοδο (π.χ. σχόλια σε κώδικα)

Αρχικές Συνθήκες (4)

- Μια αρχική συνθήκη ενεργοποιείται (ή αλλάζει) με την ενέργεια **BEGIN**
 - **BEGIN (CONDITION)**
- Μέχρι την εκτέλεση της επόμενης ενέργειας **BEGIN**, οι κανόνες με τη συγκεκριμένη αρχική συνθήκη θα είναι ενεργοί και οι κανόνες με άλλες αρχικές συνθήκες θα είναι ανενεργοί (αν είναι αποκλειστική)
- Με την **BEGIN (0)** ή την **BEGIN (INITIAL)** επιστρέφουμε στην αρχική κατάσταση, όπου μόνο κανόνες χωρίς αρχικές συνθήκες είναι ενεργοί
- Η τρέχουσα αρχική συνθήκη δίνεται μέσω του **YY_START** (ακέραια τιμή)

Αρχικές Συνθήκες (5)

Παράδειγμα Λεκτικού Αναλυτή (scanner) ο οποίος αναγνωρίζει C comments διατηρώντας παράλληλα κι ένα μετρητή για την τρέχουσα γραμμή εισόδου

```
%{
    int line_num = 1;
}%
/* definitions section */
%x comment
%%
"/*"                                BEGIN(comment) ;

<comment>[^\n]*                    /* eat anything that's not a '*' */
<comment>"*"+[^\n]*                /* eat up '*'s not followed by '/'s */
<comment>\n                        ++line_num;
<comment>"*"+"/"                   BEGIN(INITIAL) ;
```

Μεταβλητές και Συναρτήσεις (1/4)

Σύμβολο

Περιγραφή

`int yylex()`

- Η κύρια συνάρτηση του Λεκτικού Αναλυτή η οποία σαρώνει το αρχείο εισόδου για την αναγνώριση Λεκτικών Μονάδων
- Η σάρωση συνεχίζεται έως το τέλος του αρχείου εισόδου είτε έως ότου κάποια ενέργεια εκτελέσει ένα `return` statement
- Επιστρέφει έναν ακέραιο αριθμό, που συνήθως αντιστοιχεί σε μια Λεκτική Μονάδα ή 0 για `EOF`. Για συνεργασία με τον `bison` πρέπει να θέσει για κάθε Λεκτική Μονάδα την κατάλληλη τιμή στην καθολική μεταβλητή `yylval` του `bison`

`char*`
`yyltext`

Περιέχει το τμήμα της εισόδου που αντιστοιχεί στο ταίριασμα κατά την τελευταία κλήση της `yylex()`

`int yyleng`

Περιέχει το πλήθος των χαρακτήρων που περιέχονται στο `yyltext`

Μεταβλητές και Συναρτήσεις (2/4)

Σύμβολο

Περιγραφή

Δηλώνει ότι στο επόμενο ταίριασμα η Λεκτική Μονάδα πρέπει να προσαρτηθεί στην τρέχουσα τιμή της **yytext** αντί να την αντικαταστήσει

void yymore()

Παράδειγμα (input: “meta-analysis” output: “meta-meta-analysis”)

%%

```
meta-      ECHO; yymore();
analysis   ECHO;
```

Επιστρέφει τους χαρακτήρες της τρέχουσας Λεκτικής Μονάδας, εκτός από τους πρώτους n, πίσω στο ρεύμα εισόδου (input stream). Για παράδειγμα, ο ακόλουθος κώδικας για την είσοδο “foobar” θα εκτυπώσει “foobarbar”:

void yyless(int n)

%%

```
foobar      ECHO; yyless(3);
[a-z]+      ECHO;
```


Μεταβλητές και Συναρτήσεις (3/4)

Σύμβολο

Περιγραφή

`int input()`

Διαβάζει τον επόμενο χαρακτήρα από το ρεύμα εισόδου

`void unput(char c)`

Τοποθετεί τον χαρακτήρα `c` πίσω στο τρέχον ρεύμα εισόδου (θα είναι ο επόμενος χαρακτήρας που θα σαρωθεί)

`int yyterminate()`

Τερματίζει τη λειτουργία της συνάρτησης `yylex()` επιστρέφοντας 0

`int yywrap()`

Καλείται κάθε φορά που συναντάται τέλος αρχείου. Αν επιστραφεί 1, η λεκτική ανάλυση τερματίζεται, ενώ αν επιστραφεί 0 συνεχίζει

`FILE *yyin`

Το αρχείο εισόδου το οποίο σαρώνει η `yylex` για την αναγνώριση Λεκτικών Μονάδων

Μεταβλητές και Συναρτήσεις (4/4)

Σύμβολο

Περιγραφή

`void
yyrestart(FILE *f)`

Παίρνει ως όρισμα ένα δείκτη σε αρχείο και αρχικοποιεί το `yyin` για σάρωση από αυτό

`FILE* yyout`

Το αρχείο εξόδου στο οποίο γράφεται η έξοδος του Λεκτικού Αναλυτή (**ECHO** output)

Αναφορές

- Εγχειρίδιο χρήσης (manual) του FLEX, έκδοση 2.6.4
- Διαφάνειες Φροντιστηρίου Θεωρίας Υπολογισμού 2014