ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΑΘΗΝΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΡΙΚΗΣ

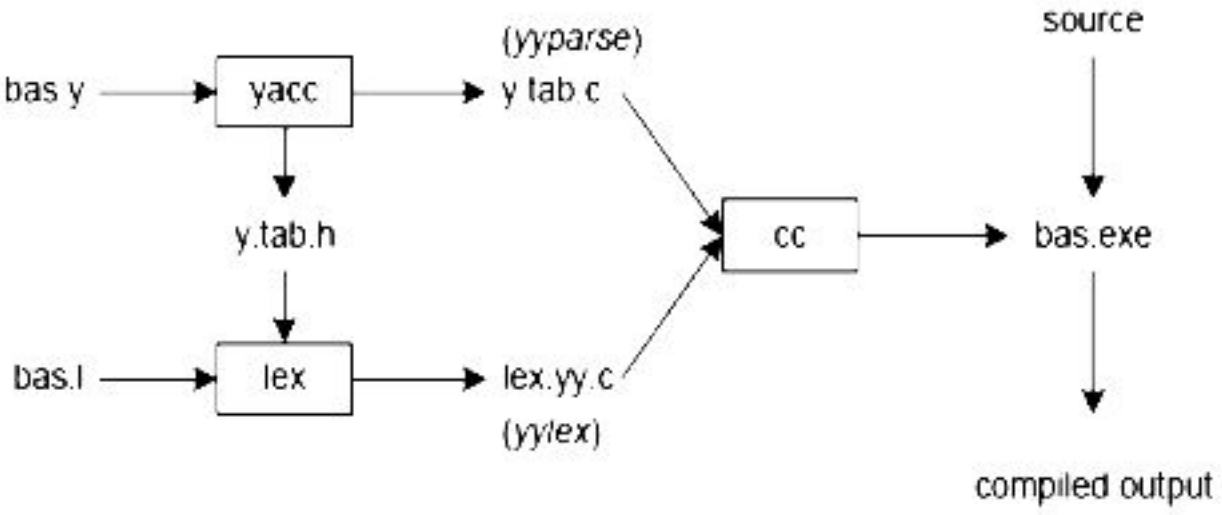
ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

ATHENS UNIVERSITY OF APPLIED SCIENCES

ΟΜΑΔΑ Ε4-2
ΑΓΓΕΛΙΚΗ ΣΦΥΡΙΔΑΚΗ
ΔΗΜΗΤΡΗΣ ΧΑΤΖΗΣ
ΠΑΡΙΣ ΛΙΖΑΙ

Περιεχόμενα

0.Εισαγωγή 3
1.Γεννήτρια Λεκτικής Ανάλυσης –FLEX 4
1.1 Γενικά για την Flex 4
1.2 Κώδικας λεκτικής ανάλυσης & επεξήγηση 5
1.2.1 Tokens 5
1.2.3 ERRORS 9
2. ΓΕΝΝΗΤΡΙΑ ΣΥΝΤΑΚΤΙΚΗΣ ΑΝΑΛΥΣΗΣ – BISON 11
2.1 ΓΕΝΙΚΑ ΓΙΑ BISON 11
2.2 ΚΩΔΙΚΑΣ ΣΥΝΤΑΚΤΙΚΗΣ ΑΝΑΛΥΣΗΣ & ΕΠΕΞΗΓΗΣΗ 12
2.3 ERRORS 22
2.4 Συναρτήσεις 22



0.Εισαγωγή

Σκοπός της εργασίας είναι η δημιουργία ενός συντακτικού αναλυτή ικανού να διαπιστώνει την ορθότητα των δομών του πηγαίου κώδικα βάσει της γραμματικής της γλώσσας.
Η είσοδος στον συντακτικό αναλυτή είναι μια ακολουθία από λεκτικές μονάδες(tokens), που παράγονται από τον λεκτικό αναλυτή μετά τον διαχωρισμό και την αναγνώριση των λεξημάτων της συμβολοσειράς εισόδου.
Οι λεκτικές μονάδες επιστρέφονται από τον λεκτικό αναλυτή έπειτα από συνεχή αιτήματα του συντακτικού αναλυτή κατά τη διάρκεια των ελέγχων που πραγματοποιεί.
Μέσα από την εργασία θα γίνει μια παρουσίαση της γεννήτριας συντακτικής ανάλυσης BISON καθώς και την αντίστοιχη γεννήτρια λεκτικής ανάλυσης FLEX.
Θα γίνει κατανοητή και λεπτομερής περιγραφή και επεξήγηση της γραμματικής τόσο της BISON όσο και της FLEX.
Ως είσοδος θα δίνεται ένας πηγαίος κώδικας mini-Python που θα περιέχει σωστές και λανθασμένες λέξεις και εκφράσεις της γλώσσας.

Πάνω απ΄ όλα όμως κύριος στόχος είναι η κατανόηση της συνεργασίας μεταξύ συντακτικού και λεκτικού αναλυτή καθώς και η απόκτηση εμπειρίας.

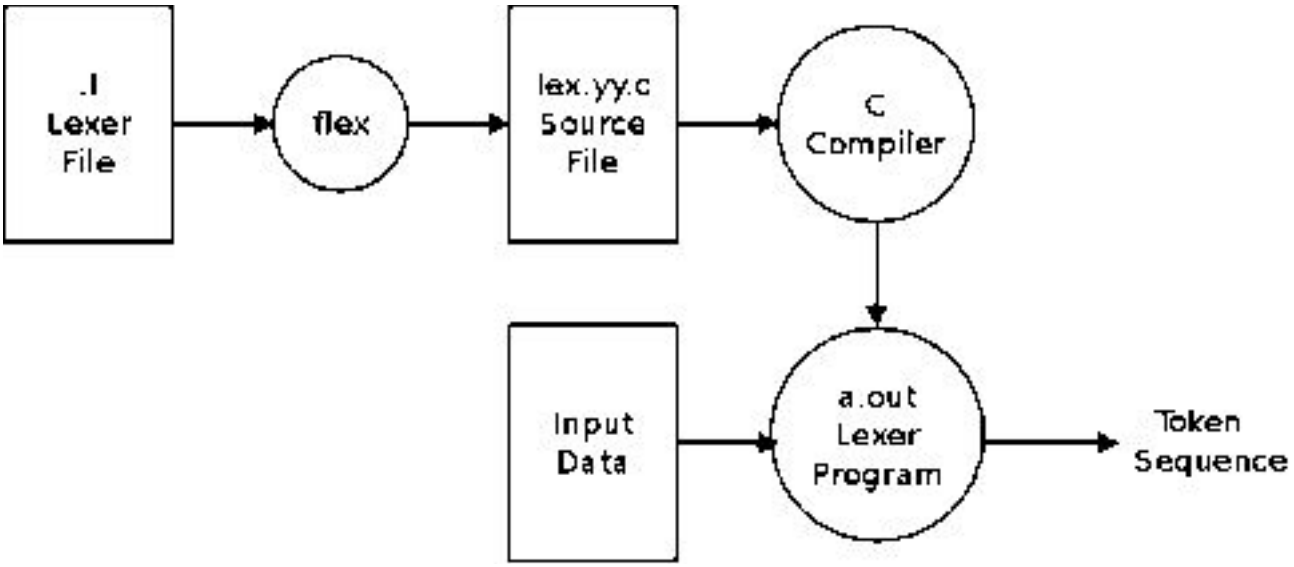
1.Γεννήτρια Λεκτικής Ανάλυσης –FLEX

1.1 Γενικά για την Flex

Η λεκτική ανάλυση είναι η πρώτη φάση ενός μεταγλωττιστή. Η κύρια λειτουργία του είναι να διαβάσει τους χαρακτήρες εισόδου, να τους χωρίσει σε λεκτήματα και να παρουσιάσει σαν έξοδο μια ακολουθία από σύμβολα της γραμματικής που αναπαριστά την γλώσσα προγραμματισμού στην οποία είναι γραμμένο το πρόγραμμα, τα οποία αντιστοιχούν στα λεκτήματα εισόδου. Εκτός από την κύρια αυτή λειτουργία, ο λεκτικός αναλυτής είναι συχνά επιφορτισμένος και με κάποιες δευτερεύουσες, πλην όμως σημαντικές λειτουργίες:

- την αφαίρεση των σχολίων
- των κενών που εμφανίζονται (κενοί χαρακτήρες, αλλαγές γραμμής, κ.λπ.)
- συσχέτιση μηνυμάτων λάθους που τυχόν παράγει ο μεταγλωττιστής με το πρόγραμμα εισόδου.

Όπως είπαμε και προηγουμένως, οι λεκτικοί αναλυτές είναι ουσιαστικά πεπερασμένα αυτόματα, τα οποία χρησιμοποιώντας κανονικές εκφράσεις προσπαθούν να ταιριάξουν κάθε σύμβολο εισόδου με κάποιους κανόνες οι οποίοι περιγράφουν την εκάστοτε γλώσσα προγραμματισμού για την οποία επιθυμούμε να φτιάξουμε τον λεκτικό αναλυτή. Για την κατασκευή αυτών των αυτομάτων, στην πράξη χρησιμοποιούμε γεννήτορες λεκτικών αναλυτών, δηλαδή προγράμματα στα οποία δίνουμε την περιγραφή της γραμματικής μας και παίρνουμε στην έξοδο το επιθυμητό αυτόματο.



1.2 Κώδικας λεκτικής ανάλυσης & επεξήγηση

1.2.1 Tokens

ΤΜΗΜΑ ΟΡΙΣΜΩΝ

INT:

```
integer     ((0+|[1-9]({digit})*))
```

Ξεκινάει με 0 μια ή περισσότερες φορές

ή ξεκινάει με ένα οποιοδήποτε ψηφίο εκτός του 0

και ακολούθως μπορεί να έχει οποιοδήποτε ψηφίο

FLOAT:

exponent [eE] {prefix}{digit}+

Ξεκινάει με E ή e

Έπειτα κάποιο πρόσημο

Ψηφίο 1 ή περισσότερες φορές

```
(({integer}\.({digit})*)|(\.({digit})+))({exponent})?
```

(0+|[1-9]([0-9])*

[0-9]

[0-9]

([eE][-+]?[0-9]+)

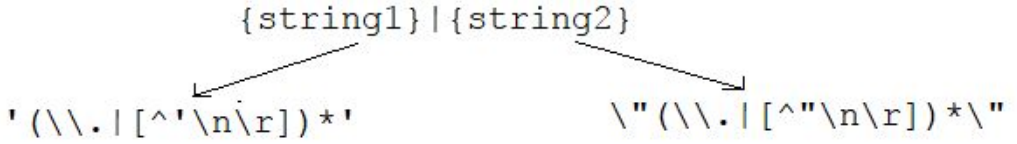
Ξεκινάει με ακέραιο συνεχίζει με τελεία

και μπορεί να ακολουθεί ψηφίο 0 ή περισσότερες φορές

ή στην περίπτωση που δεν έχει ακέραιο μέρος ξεκινάει με τελεία και περιέχει τουλάχιστον ένα ψηφίο

Επίσης μπορεί να περιέχει εκθετικό μέρος

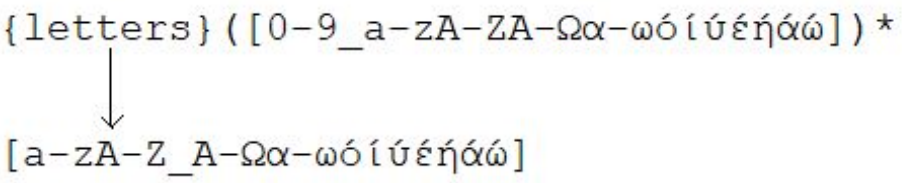
STRING:



Εντοπίζει string τα οποία μπορούν να περιέχουν οποιοδήποτε χαρακτήρα εκτός των \n 'αλλά στην περίπτωση που εντοπίζει \ μπορεί να ακολουθεί οτιδήποτε μεταξύ άλλων και το '

Εντοπίζει string τα οποία μπορούν να περιέχουν οποιοδήποτε χαρακτήρα εκτός των \n “αλλά στην περίπτωση που εντοπίζει \ μπορεί να ακολουθεί οτιδήποτε μεταξύ άλλων και το “

VAR:



Ένα γράμμα και 0 ή περισσότεροι αλφαριθμητικοί χαρακτήρες.

COMMENT:

```
(\"\"\"((\"?\"?[^"]+)+\"?\"?)?\"\"\")|{sh}
```

↓

```
(#.*[\n\r])
```

Ξεκινάει με “””
Στην περίπτωση που υπάρχει τουλάχιστον ένας χαρακτήρας μπορεί τότε να ξεκινάει με επιπλέον δύο”. Επίσης αυτό επιτρέπει την ύπαρξη ενός ή δύο “ μέσα στο περικλειόμενο κείμενο. Τέλος προβλέπει την ύπαρξη δύο “ πέραν των τριών τελικών “ εάν και μόνο εάν έχει υπάρξει κάποιος χαρακτήρας. Αντικαθιστά έτσι το ("{3,5}?([^"]+("?"?))*

ή ξεκινάει με (#) και συνεχίζει με οτιδήποτε καμία ή περισσότερες φορές, εκτός από αλλαγή γραμμής. Στη συνέχεια καταναλώνει την αλλαγή γραμμής.
(\r χρησιμεύει για ρόλους συμβατότητας. Σε κάποιες εκδόσεις η αλλαγή γραμμής γίνεται με \r\n)

```
"""correct"""
#comment
"""33"""
"""34""""
""""43"""
""""45"""""
"""""53"""
"""""55"""""

"""31"
""23"""

"""32""2

""1"3"""

"""3
1"
3"""

"""wrong"""wrong
"""31"
""23"""again

"""32""""""ll
```

- Αν εντοπισθεί μια ακολουθία χαρακτήρων που τηρούν τις προϋποθέσεις του κανόνα τότε αυξάνεται ο μετρητής (correct++)
- Όταν αναγνωρισθεί μια λεκτική μονάδα είναι αποθηκευμένη σαν χαρακτήρας. Για να επιστραφεί πρέπει να γίνει μετατροπή σε float (yylval.num = atof(yytext)) για τους ακεραίους και τους δεκαδικούς και (yylval.name=strdup(yytext)) αντιγραφή των περιεχόμενων του yytext στο yyval
- Έπειτα εκτυπώνεται στο αρχείο εξόδου (ECHO)
- Επιστρέφεται το κατάλληλο token (return)

Στην περίπτωση των σχολίων δεν χρειάζεται να επιστραφεί κάτι εφόσον δεν είναι απαραίτητα σε κάποιον γραμματικό κανόνα. Επίσης καλούμε την συνάρτηση countlines για να μετρήσει τις γραμμές που υπάρχουν στα σχόλια που επεκτείνονται σε πολλές γραμμές.

Ακολουθούν τα ATOMS της γλώσσας—Κανόνες

```
%%
"print"         {correct++;ECHO;return PRINT;}
"input"         {correct++;ECHO;return INPUT;}
"len"           {correct++;ECHO;return LEN;}
"type"          {correct++;ECHO;return TYPE;}
"sep"           {correct++;ECHO;return SEP;}
"end"           {correct++;ECHO;return END;}
[ \t]+              { }

"+"     {correct++;ECHO;return PLUS; }
"-"     {correct++;ECHO;return MINUS;}
"*"     {correct++;ECHO;return MULT;}
"/"     {correct++;ECHO;return DIV;}
"%"     {correct++;ECHO;return MOD;}
"("     {correct++;ECHO;return '(';}
")"     {correct++;ECHO;return ')';}
"**"    {correct++;ECHO;return POWER;}
"//"    {correct++;ECHO;return IDIV;}

"="     {correct++;ECHO;return EQUALS;}
"*="    {correct++;ECHO;return EMULT;}
"+="    {correct++;ECHO;return EADD;}
"-="    {correct++;ECHO;return EMINUS;}
"/="    {correct++;ECHO;return EDIV;}
"%="    {correct++;ECHO;return EMOD;}
"**="   {correct++;ECHO;return EPOWER;}
"//="   {correct++;ECHO;return EIDIV;}

"\n"    {lines++;ECHO;return NL;}
"," {correct++;ECHO;return COMMA;}

"["     {correct++;ECHO;return '[';}
"]"     {correct++;ECHO;return ']';}

"{" {correct++;ECHO;return '{';}
"}" {correct++;ECHO;return '}';}
```

Οι χαρακτήρες σκανάρονται από το αρχείο εισόδου ένας ένας, ελέγχονται και όταν η ακολουθία συμπίπτει με μια ακολουθία ορισμένη στον Flex, αναγνωρίζεται και επιστρέφεται ως το αντίστοιχο token. Π.χ. "print" {correct++; ECHO; return PRINT;}

- Το print διαβάζεται χαρακτήρας χαρακτήρας από το αρχείο εισόδου
- Αναγνωρίζεται από τον τμήμα ορισμών της FLEX
- Συμπίπτει με τον κανόνα που είναι ορισμένος (“print”)
- Τότε ο μετρητής σωστών tokens αυξάνεται (correct++)
- Γίνεται εκτύπωση στο αρχείο (ECHO)
- Και επιστρέφεται το αντίστοιχο token (PRINT)

Η ίδια διαδικασία γίνεται σε όλο το τμήμα κανόνων.
Δηλώνεται ο χαρακτήρας ή η ακολουθία χαρακτήρων. Αν εντοπισθεί, αυξάνεται ο μετρητής, εκτυπώνεται ο χαρακτήρας ή η ακολουθία στο αρχείο εξόδου και επιστρέφεται το αντίστοιχο token το οποίο αργότερα στέλνεται στον συντακτικό αναλυτή .

1.2.3 ERRORS

Όπως ακριβώς γίνεται δήλωση των ορισμών και των κανόνων για αναγνώριση των ορθών tokens έτσι γίνεται δήλωση κανόνων για πρόβλεψη και ανάνηψη των μη ορθών.

ΤΜΗΜΑ ΟΡΙΣΜΩΝ

```
string_error1   '(\\.|[^'\n\r])*
string_error2   \"(\\.|[^"\n\r])*
int_error   (0+[1-9]+)
float_error1    {int_error}\.
float_error2    {integer}?(\.[0-9]*[Ee])
var_error   [0-9]+[a-z]+
```

string_error1 '(\\.|[^'\n\r])*

Το λάθος βρίσκεται στο ότι δεν κλείνουν τα μονά αυτάκια (')

string_error2\"(\\.|[^"\n\r])*

Το λάθος βρίσκεται στο ότι δεν κλείνουν τα διπλά αυτάκια (")

int_error (0+[1-9]+)

Ένας ακέραιος δεν μπορεί να ξεκινάει με μηδέν Π.Χ. 02, 004

float_error1 {int_error}\.

Ένας float δεν μπορεί να ξεκινάει με 0

float_error2 {integer}?(\.[0-9]*[Ee])

Ένας εκθετικός δεν επιτρέπεται μετά το E, e να μην έχει κάποιο πρόσημο και έναν ακέραιο

var_error [0-9]+[a-z]+

Μια μεταβλητή δεν μπορεί να ξεκινάει με αριθμό

ΤΜΗΜΑ ΚΑΝΟΝΩΝ

```
{prefix}{int_error}                 {wrong++;ECHO;BEGIN(error);return EINTEGER;}
{float_error1}                      {wrong++;ECHO;BEGIN(error);return EEKTHETIKO1;}
{float_error2}                      {wrong++;ECHO;BEGIN(error);return EEKTHETIKO2;}
{var_error}                       {wrong++;ECHO;BEGIN(error);return EVAR;}
({string_error1}|{string_error2})           {wrong++;ECHO;BEGIN(error);countlines();return ESTRING;}
<error>[ \t\n]                      {wrong++;ECHO;BEGIN(0);countlines();}
<error>.                          {wrong++;ECHO;BEGIN(0);countlines();}
%%
```

***Τα λάθη είναι διατυπωμένα στο τμήμα ορισμών*

Η παρακάτω γραμμή χρήζει περεταίρω εξήγησης

{wrong++;ECHO;BEGIN(error);return EINTEGER;}

Μόλις αναγνωρισθεί το λάθος:

- Αυξάνεται ο μετρητής λανθασμένων λεξημάτων.
- Εκτυπώνεται το λάθος στο αρχείο εξόδου
- Η BEGIN(error) χρησιμοποιείται ώστε όταν ανιχνευθεί το λάθος να καταναλώνει τους χαρακτήρες ωσότου βρει ένα delimiter ώστε να ξεκινήσει ξανά τον έλεγχο
- Και τέλος επιστρέφεται το κατάλληλο token

2. ΓΕΝΝΗΤΡΙΑ ΣΥΝΤΑΚΤΙΚΗΣ ΑΝΑΛΥΣΗΣ – BISON

2.1 ΓΕΝΙΚΑ ΓΙΑ BISON

Στη φάση της συντακτικής ανάλυσης ελέγχεται αν το αρχικό πρόγραμμα ανήκει στη γλώσσα της οποίας η σύνταξη ορίζεται από μια δεδομένη γραμματική χωρίς
συμφραζόμενα. Η ακολουθία λεκτικών μονάδων που προκύπτει από την λεκτική
ανάλυση διαβάζεται στη φάση αυτή, και οι λεκτικές μονάδες ομαδοποιούνται σε συντακτικές μονάδες με βάση την παραπάνω γραμματική. Παράλληλα κατασκευάζεται το συντακτικό δέντρο(syntax tree) που απεικονίζει τη συντακτική δομή του αρχικού προγράμματος. Ανάλογα με τη σχεδίαση των επόμενων φάσεων ενδιάμεσου κώδικα, οπότε και αποθηκεύεται για μεταγενέστερη χρήση.

Όπως αναφέρθηκε παραπάνω, η είσοδος ςνός συντακτικού αναλυτή είναι το αρχικό πρόγραμμα με τη μορφή μιας ακολουθίας λεκτικών μονάδων και η έξοδος του, ένα συντακτικό δέντρο που αντιστοιχεί σε αυτό το πρόγραμμα ή μια ένδειξη ότι το αρχικό πρόγραμμα δεν είναι συντακτικά ορθό. Στη πράξη όμως ο συντακτικός αναλυτής συνεργάζεται στενά με τις επόμενες φάσεις της μεταγλώττισης και το συντακτικό δέντρο δεν είναι ορατό ως έξοδος αυτού του τμήματος.

Συνοψίζοντας ο συντακτικός αναλυτής:

- Υλοποιεί τη συντακτική ανάλυση μιας συγκεκριμένης γλώσσας
- Δέχεται ως είσοδο ένα πρόγραμμα με τη μορφή ακολουθίας λεκτικών μονάδων
- Ελέγχει αν το πρόγραμμα είναι σύμφωνο με τη γραμματική της γλώσσας
  που υλοποιεί (αν ανήκει στη συγκεκριμένη γλώσσα)
- Σε περίπτωση συντακτικού λάθους ενημερώνει το χρήστη
- Παράγει το συντακτικό δέντρο που αντιστοιχεί στην ακολουθία εισόδου

2.2 ΚΩΔΙΚΑΣ ΣΥΝΤΑΚΤΙΚΗΣ ΑΝΑΛΥΣΗΣ & ΕΠΕΞΗΓΗΣΗ

Μερικές σημαντικές πληροφορίες

Error-verbose

Χρησιμοποιείται για να πάρουμε πιο αναλυτικά μηνύματα λάθους στην κλήση της yyerror

Union

Ορίζει τους τύπους που μπορούν να πάρουν τα σύμβολα (τερματικά και μη)

```
{
double num;
char*  name;
}
```

```
%type  <num>  line exp met input len content
%type  <name> print print2 type lista string_con errors
```

ΠΕΡΙΟΧΗ ΔΗΛΩΣΗΣ ΤΕΡΜΑΤΙΚΩΝ ΟΡΩΝ

Εδώ γίνεται δήλωση όλων το tokens που επιστρέφει ο λεκτικός αναλυτής εκτός των atoms

```
%token <name> PRINT LEN SEMI INPUT TYPE SEP END COMMA
%token <num>  INT FLOAT
%token <name> STRING VAR
%token PLUS MINUS MULT DIV MOD POWER IDIV NL
%token EQUALS EMULT EADD EMINUS EDIV EMOD EPOWER EIDIV
```

ΠΕΡΙΟΧΗ ΔΗΛΩΣΗΣ ΠΡΟΤΕΡΑΙΟΤΗΤΩΝ

%left , %right

> Ορίζουν την προτεραιότητα στα token που ακολουθούν στη γραμμή με αυξανόμενη προτεραιότητα από πάνω προς τα κάτω

Π.Χ.

%left add, sub
Το left σημαίνει ότι έχουμε αριστερή προτεραιότητα
Το 1+2-3 σημαίνει (1+2) -3

%right mult, div
Το right σημαίνει ότι έχουμε δεξιά προτεραιότητα.
Το x=3+5 σημαίνει ότι πρώτα θα γίνουν οι πράξεις δεξιά του = και μετά η εκχώρηση τιμής

```
%right EQUALS
%right EMULT EADD EMINUS EDIV EMOD EPOWER EIDIV
%left  MINUS PLUS
%left  MULT DIV MOD IDIV
%right POWER
%left  '(' ')'
```

ΠΕΡΙΟΧΗ ΚΑΝΟΝΩΝ

Η αρχή του προγράμματος δηλώνεται με την εντολή %start

ΑΡΧΗ

```
start:    /* empty */
        | start line
    ;
```

Το αρχείο εισόδου μπορεί να ξεκινάει με κενή γραμμή

Μπορεί να είναι πάλι ο κανόνας κενή γραμμή(καμία, μια ή περισσότερες φορές) ή line που αναλύεται παρακάτω

Γίνεται αισθητή η λειτουργία της αναδρομής.
Ο πρώτος κανόνας εφαρμόζεται στο δεύτερο και έτσι όταν κληθεί η start line το πρόγραμμα θα διαβάσει start και θα εξετάσει τι μπορεί να είναι. Δηλαδή κενό ή start line και ξανά το ίδιο μεχρι να εκπληρωθεί ο κανόνας!!!!

ΕΝΤΟΛΕΣ ΠΟΥ ΜΠΟΡΟΥΝ ΝΑ ΣΤΑΘΟΥΝ ΑΥΤΟΝΟΜΑ

```
line:     NL          {}
        | print NL    {correct_exp++;}
    | met NL      {correct_exp++;}
    | error NL    {wrong_expr++; flag=1;yyerrok;}
    ;
```

Μπορεί να είναι new line (το NL είναι token το οποίο επιστρέφει ο FLEX)

Μπορεί να είναι print νέα γραμμή. Αυξάνεται ο μετρητής σωστών εκφράσεων

Εκχώρηση τιμής σε μεταβλητή και νέα γραμμή. Αύξηση του μετρητή σωστών εκφράσεων

Λάθος και νέα γραμμή. Εδώ αυξάνεται ο μετρητής λανθασμένων εκφράσεων. Το flag γίνεται ένα ως ένδειξη του λάθους και καλείται η yyerrok για να διαχειριστεί το πρόβλημα.

Η PRINT2

```
print2:   content              {}
    | print2 COMMA content      {}
    ;
```

Η print2 μπορεί να είναι content δηλαδή int, float, string, var

Εδώ υπάρχει πάλι αναδρομή. Μέσα στον κανόνα γίνεται χρήση του ίδιου του κανόνα. Αυτό σημαίνει πως εκτελείται στον εαυτό του. Άρα μπορεί να ξεκινάει με content κόμμα content ή μόνο content

Η PRINT

```
print:    PRINT '(' print2 COMMA SEP EQUALS STRING COMMA END EQUALS STRING ')' {}
    | PRINT '(' print2 COMMA END EQUALS STRING ')' {}
    | PRINT '(' print2 COMMA SEP EQUALS STRING ')' {}
    | PRINT '(' print2 ')'                  {}
    | PRINT '(' SyntaxError                 {correct_exp--;flag=1;}
    ;


print2:   print3          {}
    | print2 COMMA print3   {}
    ;

print3:   content         {}
    | type          {}
    ;
```

Η print ξεκινάει με PRINT και μέσα στις παρενθέσεις μπορεί να έχει print2 που αναλύθηκε παραπάνω, έπειτα κόμμα, μετά sep = με string , και = κάποιο string

Π.Χ. print(1,2,3,4,sep='bla', end='&')

Από την print2

Όμοια η 2η γραμμή Π.Χ. print(bla, 3, end='abcd')
3η γραμμή Π.Χ. print (abla, 3,4,5, x4, sep='*')
4η γραμμήο κανόνας print2
5η γραμμή να είναι λάθος οπότε και μειώνεται ο μετρητής σωστών εκφράσεων.

Η INPUT

```
input:    INPUT  '(' ')'    {}
    | INPUT  '(' STRING ')' {}
    | INPUT  '(' errors {correct_exp--;flag=1;}
    ;
```

Ξεκινάει με input και κενές παρενθέσεις

INPUT και ένα string μέσα στις παρενθέσεις

Κάποιο λάθος, οπότε μειώνεται ο μετρητής σωστών εκφράσεων επειδή πριν τον αυξήσαμε

```
>>>x=input()
4
>>> x
'4'
>>> x=input('Δώσε τιμή για το x = ')
Δώσε τιμή για το x = 4444
>>> x
'4444'
```

Η LEN

```
len:      LEN '(' VAR ')'    {}
    | LEN '(' '[' lista ']' ')'{}
    | LEN  '(' errors   {correct_exp--;flag=1;}

    ;
```

Αρχίζει με LEN και μέσα στις παρενθέσεις μπορεί να έχει κάποια μεταβλητή

Να αρχιζει με LEN και μέσα στις παρενθέσεις να περιέχεται μια λίστα(αναλύεται παρακάτω)

Να είναι λάθος οπότε πρέπει να μειωθεί ο μετρητής σωστών εκφράσεων επειδή πριν τον ωαυξήσαμε

```
y='My first name is Κώστας '
>>> len(y)
24
```

ΛΙΣΤΑ

```
lista:   '[' ']'          {}
     | content            {}
     |'[' lista ']'       {}
     | lista COMMA lista {}
     ;
```

Μπορεί να είναι κενή. Μια λίστα μπορεί να είναι κενή. Υπάρχει ένας πολύ βασικός λόγος για τον οποίο πρέπει να διαθέσουμε ξεχωριστό κανόνα ('[' ']') και όχι απλά το κενό/τίποτα. Σε αυτή την περίπτωση θα επιτρέπαμε στον συντακτικό αναλυτή να δεχτεί το [α ,β ,]

Να είναι ένας int, float, string ή var

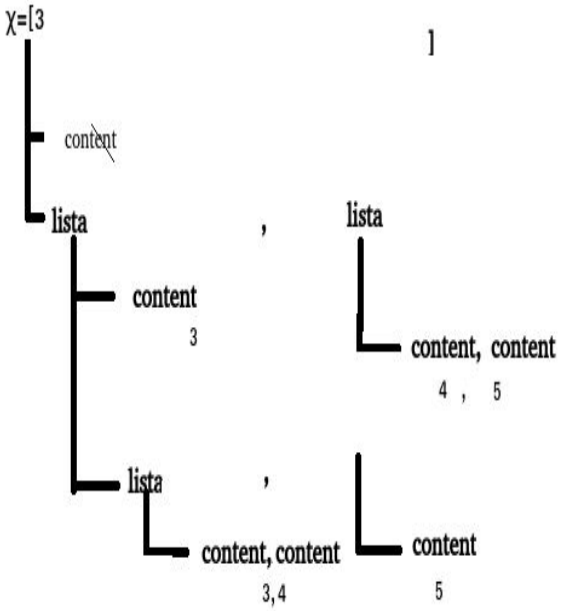
Χάρη στην αναδρομή να είναι είτε πάλι η κενή λίστα ή να περιέχει κάποιο content

Είτε να είναι οποιαδήποτε από τις άλλες 3 περιπτώσεις (,) και πάλι μια από τις προηγούμενες 3 περιπτώσεις

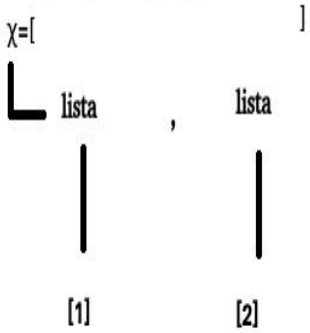
Στην δίπλα εικόνα παρατηρούμε ότι η πρώτη λίστα προκύπτει με δύο τρόπους από το bison και γι αυτό μας εμφανίζει μήνυμα λάθους. Εάν όμως εξετάσουμε την δεύτερη λίστα κατανοούμε την ανάγκη για την ύπαρξη αναδρομής αριστερά και δεξιά του κόμματος.

Λόγως εμφάνισης warning reduce/reduce

Έστω η λιστα x=[3,4,5]



έστω τώρα η λίστα χ=[[1],[2]]




```
>>> pinax = [A, 'my', ' ', 'good friend']
>>> lista = [1, A, 2, AB, 'Hello']
>>> pinax = ['A','B','Γ',['x','y','z']]
>>> pinax = [['A','B],[1,2]]
>>> lista = []
>>> lista = [a,b,c,[]]
```

CONCATENATION

```
string_con:STRING       {}
    |  STRING PLUS string_con {}
    ;
```

Μπορεί να είναι ένα string όπως αυτό ορίζεται

Μέσω αναδρομής μπορεί ένα string να προστεθεί σε άλλα string(όσα δηλώσει ο χρήστης δημιουργώντας ένα νέο string)

```
>>> x='My first name is' + ' ' + 'Κώστας'
>>> print(x)
'My first name is Κώστας'
>>> print(x+"and my age is"+3)
'My first name is Κώστας and my age is 3'
```

ΠΛΕΙΑΔΑ

```
pleiada:  content           {}
    |'(' pleiada  COMMA content')'  {}
    | pleiada COMMA content     {}
    ;
```

Μπορεί να είναι int, float, string, var

Μέσω αναδρομής να μέσα σε παρενθέσεις ξεκινάει με content[να επαναλαμβάνεται όσες φορές είναι] μετά (,) και ξανά content

Το ίδιο με το προηγούμενο χωρίς παρενθέσεις

```
x=('a',2,'b')
y=(2,'a')
v=(a,(b,c),d)
```

Μια πλειάδα περιέχει μέσα σε παρενθέσεις ένα πλήθος στοιχείων που χωρίζονται μεταξύ τους με κόμμα. Το πλήθος των στοιχείων πρέπει απαραιτήτως να είναι τουλάχιστον 2 το οποίο προέκυψε λογικά στο x=(3). Έτσι "ρωτήσαμε" έναν interpreter τον τύπο του χ και προέκυψε ότι είναι ακέραιος

DIMENSION

```
dimension: '['exp ']'       {}
    |  dimension '['exp ']' {}
    ;
```

Μπορεί να περιέχει μια τιμή exp μέσα που αναλύεται παρακάτω. Άρα πρόκειται για μονοδιάστατο πίνακα

Με αναδρομή ο πίνακας γίνεται πολλών διαστάσεων. Όσες βάλει ο χρήστης

```
>>> x=pinax[0]
>>> pinax [0][1]='B'
>>> name="GEORGE"
>>> print(name[len(name)-1])
>>> 'E'
>>> x=pinax[i-1][j*1+1][i+j]
```

TYPE

```
type:     TYPE '(' VAR ')'  {}
    | TYPE '(' errors   {correct_exp--;flag=1;}
    ;
```

Ξεκινάει με TYPE και μεέσα στις παρενθέσεις μπαίνει ια μεταβλητή και η type επιστρέφει πίσω τον τύπο της.

Π.Χ. x="ancd" >>>type (x) <class 'str'>

Μπορεί να είναι λάθος οπότε μειώνεται ο μετρητής των σωστών εκφράσεων επειδή πριν τον αυξήσαμε

```
>>> number=3
>>> print(type(number))
<class 'int'>

>>> name="EYTYXIA"
>>> print(type(name))
<class 'str'>
```

```
>>> y=(2,'a')
>>> print(type(y))
<class 'tumple'>

>>> y=(2)
>>> print(type(y))
<class 'int'>
```

ΜΕΤΑΒΛΗΤΕΣ ΚΑΙ ΕΚΧΩΡΗΣΕΙΣ

```
/*ana8esh timhs*/
met:  VAR EQUALS input{}
    | VAR EQUALS VAR dimension      {}
    | VAR EQUALS '['lista']'        {}
    | VAR EQUALS '[' ']'            {}
    | VAR dimension EQUALS exp      {}
    | VAR EQUALS '('pleiada COMMA pleiada ')'{}
    | VAR EQUALS string_con         {}
    | VAR EQUALS exp                {if (!flag) fprintf(yyout,"%.2f\n",$3);}
    | VAR EMULT exp                 {}
    | VAR EADD exp                  {}
    | VAR EMINUS exp                {}
    | VAR EDIV exp                  {}
    | VAR EMOD exp                  {}
    | VAR EPOWER exp                {}
    | VAR EIDIV exp                 {}
    | SyntaxError                   {wrong_expr++; flag=1;}
    ;
```

Στην προκειμένη περίπτωση υπάρχουν πολλές πιθανότητες. Μπορεί στη μεταβλητή να εκχωρηθεί:

- Ένα input
- να εκχωρήσει μια τιμή ενός πίνακα
- Μια λίστα
- Μια κενή λίστα
- Μια πλειάδα
- Ένα string_con
- Μια τιμή εκτός και αν υπάρχει λάθος όπου το flag=1. Πρακτικά αυτό συμβαίνει όταν έχει βρεθεί λάθος διαίρεσης με 0.
- Μπορεί να γίνουν πράξεις παράλληλα με την εκχώρηση (*=,+=,/=,...)

ΑΡΙΘΜΗΤΙΚΕΣ ΕΚΦΡΑΣΕΙΣ

```
/*ari8mhtikes ekfraseis*/
exp:  INT          {$$=$1;}
    | FLOAT          {$$=$1;                }
    | MINUS exp      {$$=-$2;               }
    | VAR            {}
    | len            {}
    | SyntaxError    {correct_exp--;flag=1; }
    | exp PLUS exp   { $$ = $1 + $3;        }
    | exp MINUS exp  { $$ = $1 - $3;        }
    | exp MULT exp   { $$ = $1 * $3;        }
    | '(' exp ')'    { $$ = $2;             }
    | exp DIV exp    { if(!$3) {fprintf(yyout,"division by 0 not allowed\n");flag=1;}else $$ = $1 / $3;                    }
    | exp MOD exp    { if(!$3) {fprintf(yyout,"division by 0 not allowed\n");flag=1;}else {int i=$1/$3; i=$1-i*$3; $$=i;}  }
    | exp POWER exp  { int i,pow=1;for(i=0;i<$3;i++) {pow*=$1;} $$=pow;                                                     }
    | exp IDIV exp   { if(!$3) {fprintf(yyout,"division by 0 not allowed\n");flag=1;}else $$=(int)($1/ $3);               }
   ;
```

Ένα exp είναι ένας αναδρομικός με τερματικός κανόνας, έτσι μια έκφραση τέτοιου τύπου, μπορεί να αποτελείται από πολλά διαφορετικά αναδρομικά σκέλη. Άξια σημασίας αποτελούν τα, MINUS exp που καλύπτουν την περίπτωση αρνητικού πρόσημου, len όπου βρίσκει το μήκος μιας συμβολοσειράς και αναλύσαμε παραπάνω, το SyntaxError που προβλέπει την λανθασμένη σύνταξη ενός cont και τέλος ο κώδικας που έχουν οι τελεστές /,%,//. Η if ελέγχει εάν ο διαιρέτης είναι αρνητικός ώστε να προλάβει ένα τόσο σοβαρό λάθος. Εμφανίζει σχετικό μήνυμα λάθους και θέτει το flag =1 ώστε ο πατέρας μη τερματικός όρος να μην κάνει την τελική ανάθεση.

```
>>> x=20+2
22
>>> x=20-2
18
>>> x=10*2
20
>>> x=20/2
10.0
```

```
>>> x=20/2+5
15.0
>>> x=20//3
6
>>> x=20%3
2
>>> x=20//3+20%3
8
```

```
>>> x=(20//3)*3+20%3
20
>>> x=2 + -5
-3
>>> x=2**2
4
>>> x=20//0
division by 0 not allowed
>>> x=len("i do")
4
```

2.3 ERRORS

```
errors:   EINTEGER      {wrong_expr++;fprintf (yyout,"\tline:%d\t integer starts with 0 \n", lines);flag=0;}
    | EEKTHETIKO1       {wrong_expr++;fprintf (yyout,"\tline:%d\t float starts with 0 \n", lines);flag=0;}
    | EEKTHETIKO2       {wrong_expr++;fprintf (yyout,"\tline:%d\t float ends incorrectly \n", lines);flag=0;}
    | EVAR          {wrong_expr++;fprintf (yyout,"\tline:%d\t var starts with digit \n", lines);flag=0;}
    | ESTRING       {wrong_expr++;fprintf (yyout,"\tline:%d\t string does not end \n", lines);}
    ;
```

Η ανίχνευση ενός λάθους γίνεται σύμφωνα με τους κανόνες που έχουν οριστεί στο ΛΑ
Όταν ανιχνεύεται ένα error αρχικά αυξάνεται ο μετρητής των εσφαλμένων εκφράσεων.
Έπειτα γίνεται εκτύπωση στο αρχείο εξόδου των παρακάτω στοιχείων:

- Γραμμή
- Τύπος λάθους(ανάλογα με το λάθος)

2.4 ΣΥΝΑΡΤΗΣΕΙΣ

int yyerror(const char* s);

Περιέχει μια ρουτίνα διαχείρισης λεκτικών σφαλμάτων, την yyerror, η οποία καλείται όταν μια ακολουθία token της συμβολοσειράς εισόδου δεν μπορεί να αποδοθεί σε κανέναν κανόνα ή δεν έχουμε καταφέρει να ανανήψουμε όλα τα πιθανά σφάλματα. Η ρουτίνα αντιμετωπίζει το πρόβλημα με τη μέθοδο του πανικού τυπώνοντας κατάλληλο διευκρινιστικό μήνυμα.

Void countlines();

Για την καταμέτρηση των γραμμών στο αρχείο, κατέστη αναγκαία η δημιουργία μιας βοηθητικής συνάρτησης countlines(). Η συνάρτηση ελέγχει έναν έναν όλους του χαρακτήρες του λεξήματος αναζητώντας για \n ή \r, χρησιμοποιώντας τις έτοιμες συναρτήσεις yyleng(Το μέγεθος του λεξήματος που ελέγχεται) και yytext(Δείκτης προς το λέξημα που διερευνάται).