Εργασία Μεταγλωττιστών

Τμήμα Α2

Ομάδα 15

10 Απριλίου 2021

Ομάδα 15

Αναλυτικά τα μέλη:

Διονύσης Νικολόπουλος ΑΜ : 18390126

Αθανάσιος Αναγνωστόπουλος ΑΜ : 18390043

Αριστείδης Αναγνωστόπουλος ΑΜ : 16124

Σπυρίδων Φλώρος ΑΜ : 141084

Αναλυτικά οι ρόλοι:

Γενικός Συντονιστής: Διονύσης Νικολόπουλος

Υπεύθυνος Τμήματος Εργασίας Α3: *Αθανάσιος Αναγνωστόπουλος*

Εκτελέσεις παραδειγμάτων : Διονύσης Νικολόπουλος , Αριστείδης Αναγνωστόπουλος , Σπυρίδων Φλώρος, *Αθανάσιος Αναγνωστόπουλος*

USERNAMES στο Github

Διονύσης Νικολόπουλος : jiud

Θανάσης Αναγνωστόπουλος : ThanasisAnagno

Αριστείδης Αναγνωστόπουλος : Aris-Anag

Σπυρίδων Φλώρος : spirosfl

Περιεχόμενα

Εισαγωγή

Τι πραγματεύεται η εργασία.

Στο συγκεκριμένο εργαστηριακό μέρος ασχοληθήκαμε γύρω από μια γεννήτρια λεκτικών αναλυτών που ονομάζεται FLEX.

Τι είναι το FLEX ;

Το FLEX όπως προαναφέραμε είναι μια γεννήτρια λεκτικών αναλυτών, δηλαδή ένα εργαλείο που με την χρήση του βοηθάει τον προγραμματιστεί να δημιουργήσει με μεγαλύτερη ευκολία τον δικό του λεκτικό αναλυτή.

Σε ποια γλώσσα βασίζεται;

Το FLEX δημιουργήθηκε και βασίστηκε κατά κύριο βαθμό μέσω της γλώσσας C διατηρώντας μερικές ιδιότητές της, όμως χρησιμοποιεί κανονικές εκφράσεις για την αναγνώριση των λεκτικών μονάδων.

Πως δημιουργείται ο Λεκτικός Αναλυτής;

Αρχικά υπάρχει ένα πηγαίο πρόγραμμα μέσω LEX με την κατάληξη .l. Το συγκεκριμένο πρόγραμμα, το μεταγλωττίζουμε μέσω του LEX δημιουργώντας ένα αρχείο το οποίο μπορούμε πλέον και το μεταγλωττίζουμε μέσω του μεταγλωττιστή της C. Δημιουργώντας ένα κατάλληλα εκτελέσιμο αρχείο. Στο συγκεκριμένο αρχείο μετά την εισαγωγή δεδομένων παρατηρούμε τι αποτελέσματα μας έχουν προβληθεί και άμα έχει αναγνωρίσει σωστά τις λεκτικές μονάδες που του έχουμε τοποθετήσει.

Σελίδα στο Github

Έχουμε δημιουργήσει μια σελίδα που βρίσκεται στο Github, μέσω εκείνης μπορέσαμε να έχουμε όλοι μας την δυνατότητα να μοιράσουμε την πληροφορία και την πορεία που είχαμε πάνω στην εργασία μας. Μπορείτε να το επισκεφτείτε στο link από κάτω:

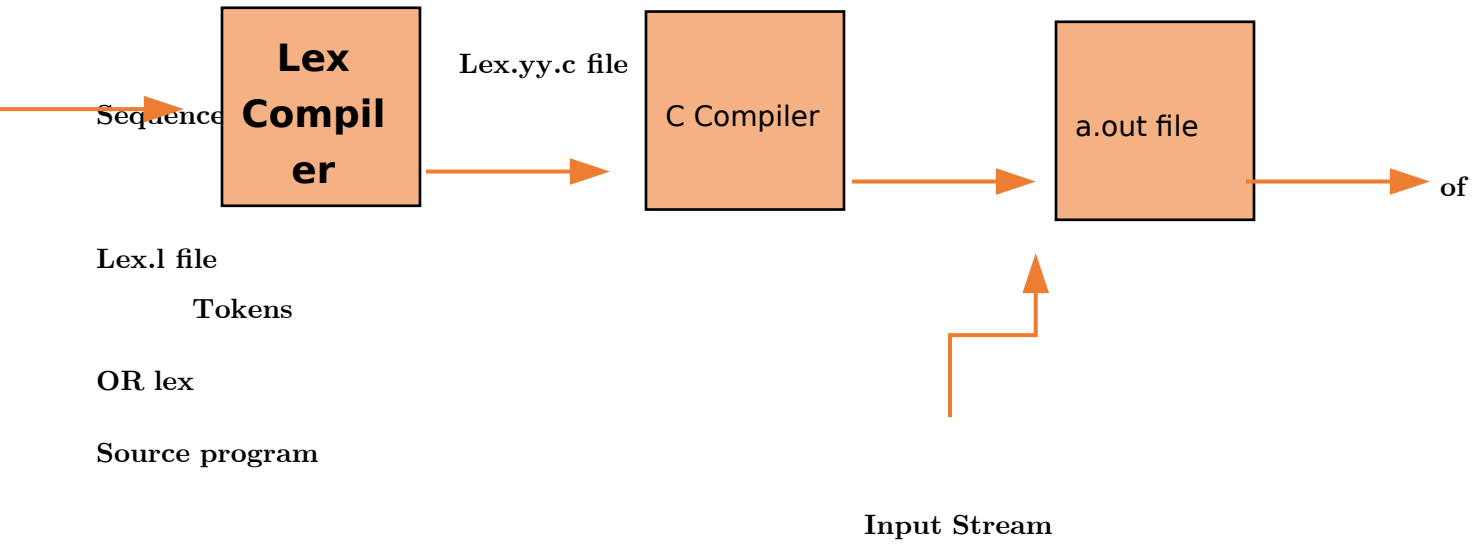
GitHub - jiud/uni-C-Analyser: A compiler for a custom version of C, uni-C.

Τι θα παρατεθεί στην εργασία

Το κύριο θέμα που λαμβάνει μέρος στην συγκεκριμένη εργασία, είναι η κατανόηση της χρήσης του λεκτικού αναλυτή FLEX καθώς και η ορθή λειτουργία του.

Στο συγκεκριμένο PDF θα παραθέσουμε αρχικά μέσω διαγραμματικής προσέγγισης την δημιουργία, μέσω απλών βημάτων, του Λεκτικού Αναλυτή LEX. Στην συνέχεια θα παραθέσουμε την δομή που διακατέχει τον λεκτικό μας αναλυτή LEX. Αφετέρου θα τεκμηριώσουμε προβάλλοντας όλα τα βοηθήματα που οδήγησαν στην δημιουργία του Λεκτικού μας Αναλυτή.

ΔΙΑΓΡΑΜΜΑΤΙΚΗ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ ΛΕΚΤΙΚΟΥ ΑΝΑΛΥΤΗ



Όπως παρατηρούμε. στο διάγραμμά μας υπάρχει το FLEX(Lex compiler) , ο compiler της C (gcc) , καθώς και το εκτελέσιμο αρχείο που παράγεται a.out file. Εμείς αφού έχουμε φτιάξει τον πηγαίο μας κώδικα με την κατάλληλη κατάληξη (.l), δηλαδή ένα πρόγραμμα FLEX(lex), το μεταγλωττίζουμε μέσω του Lex compiler για να διαπιστώσουμε άμα πληροί της προϋπόθεσης του και δεν παρατηρούνται λάθη σε σχέση με τον κώδικα που

έχουμε γράψει μέσα του. Σε περίπτωση που παρατηρηθούν, μας προβάλλει το σημείο που υπάρχει το συγκεκριμένο λάθος, για να το διορθώσουμε και να ξανατρέξουμε την διαδικασία από την αρχή, μέχρις ότου δεν υπάρχουν πλέον λάθη. Εν συνεχεία, μόλις μεταγλωττιστεί ορθά μας παράγει ένα αρχείο που έχει την κατάληξη c (παράδειγμα Lex.yy.c). Το συγκεκριμένο c αρχείο το δίνουμε σαν είσοδο στον compiler της c (gcc) και μέσω αυτού έχοντας κάνει την επιβεβαίωση για τη μη ύπαρξη λαθών μας παράγει το εκτελέσιμο αρχείο μας. Το εκτελέσιμο αρχείο όπως γνωρίζουμε πρέπει να περιλαμβάνει και μέσα του την main() ,την οποία έχουμε συμπεριλάβει στο (.l) αρχείο μας, για να μπορέσει να λειτουργήσει αυτόματα. Αν το αρχείο δεν περιλαμβάνει main για να το χρησιμοποιήσουμε για την χρήση σε σχέση με άλλο πρόγραμμα συμπεριλαμβάνουμε την οδηγία "#include lex.yy.c" που είναι το όνομα του αρχείου μας. Μόλις παρατηρήσουμε πως όλα τα βήματα έχουν προχωρήσει, τοποθετούμε τιμές σαν είσοδο στο συγκεκριμένο αρχείο, απαντώντας μας με συγκεκριμένες πληροφορίες στην έξοδο, δηλαδή μέσω των tokens που έχουμε καταχωρήσει στον πίνακα τους. Με αυτόν τον τρόπο ορίζεται η γενική λειτουργία , της γεννήτριας λεκτικών αναλυτών FLEX.

ΔΟΜΗ ΕΚΤΕΛΕΣΙΜΟΥ ΚΩΔΙΚΑ

Η δομή που περιέχουν τα προγράμματα μας με την κατάληξη του FLEX(.l) έχουν διάφορα δομικά χαρακτηριστικά. Αρχικά , υπάρχει το πεδίο του ορισμού του, μετά υπάρχει το πεδίο των κανόνων, καθώς και το πεδίο που γράφει κώδικα ο χρήστης, το οποίο είναι και προαιρετικό. Για να μπορέσουμε να ξεχωρίσουμε από ποιο σημείο αρχίζουν να ορίζονται τα συγκεκριμένα σημεία, παρατηρούμε πως μεταξύ τους υπάρχει το σύμβολο του '%%' κάνοντας ορατή την αλλαγή που παρατηρείτε ανάμεσα στις ενότητες.

Όσο αναφορά το πεδίο που βρίσκονται οι ορισμοί, έχουμε δύο υποτμήματα. Στο πρώτο υποτμήμα τοποθετούμε κώδικα της γλώσσας C , ο οποίος μέσα μπορεί να έχει για παράδειγμα βιβλιοθήκες της C , να αρχικοποιεί κατάλληλες μεταβλητές άμα είναι αναγκαίο καθώς και άλλα βασικά πράγματα που υπάρχουν στον κόσμο της C. Δηλαδή σε εκείνο το σημείο παρατηρούμε κομμάτια κώδικα που δεν ανήκουν σε κάποια συγκεκριμένη συνάρτηση και απλός είναι αναγκαία στο παραγόμενο αρχείο του λεκτικού μας αναλυτή. Επίσης το συγκεκριμένο τμήμα το ξεχωρίζουμε μέσα στην ακολουθία των χαρακτήρων % { όπου και αρχίζουν , %} όπου και τελειώνουν. Στο δεύτερο υποτμήμα, το οποίο είναι προαιρετικό μπορούμε να ορίσουμε μέσα σε ορισμούς συγκεκριμένων ονομάτων.

Το δεύτερο πεδίο ορίζει τους κανόνες. Οι κανόνες από την μια πλευρά εμπεριέχουν κανονικές εκφράσεις, ενώ από την άλλη πλευρά βρίσκεται κώδικας C, που θα εκτελεστεί μόνο αν υπάρξει σύνδεση(match) στην είσοδο μαζί με τον συγκεκριμένο κανόνα που έχουμε δημιουργήσει. Δηλαδή μέσω του ελέγχου σε κάθε στοιχείο παρατηρούν άμα το έχουν αναγνωρίσει για να το επισημάνουν στην συνέχεια.

Στο τρίτο πεδίο που βρίσκεται ο κώδικας του χρήστη τοποθετούμε όλο τον κώδικα που έχουμε γράψει στην C, έχοντας το εκτελέσιμο αρχείο που εμπεριέχει μια main() για να μπορέσει να λειτουργήσει και αυτόματα. Ιδιαίτερα στο συγκεκριμένο πεδίο, παρατηρούμε πολλές συναρτήσεις μέσω των οποίων καλούμε ένα μεγάλο μέρος των κανόνων, για να κάνουμε πιο εύκολη την χρήση τους.

Σχολιασμός Κώδικα “Uni-c-analyzer.l”

Πεδίο ορισμού (Ά μέρος)

Ξεκινάμε το αρχείο μας με την επιλογή του %option noyywrap . Ειδικότερα η συνάρτηση yywrap καλείτε μετά την αναγνώριση ενός ολόκληρου αρχείο εισόδου, δηλαδή μετά την ανάγνωση του EOF. Αν έχουμε παραπάνω από ένα αρχεία σαν είσοδο. Η συγκεκριμένη επιλογή ενημερώνει το αρχείο μας πως δεν υπάρχουν παραπάνω αρχεία οπότε δεν χρειαζόμαστε για κάποιο λόγω την ύπαρξη της yywrap.Οπότε την προσθέτουμε και την αγνοεί.

Τα δεδομένα που βρίσκονται μέσα στις ‘%{‘ είναι διάφορες βιβλιοθήκες καθώς και μεταβλητές που ορίζονται στο πρόγραμμά μας και μεταφέρονται ανεπηρέαστα μέσα στο αρχείο C που έχουμε δημιουργήσει στο FLEX. Το συγκεκριμένο μέρος του κώδικα ορίζεται στο πεδίο του ορισμού.

```
%option noyywrap
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "token.h"
int line = 1;
%}
```

Πεδίο ορισμού (Β μέρος)

Στο συγκεκριμένο μέρος του κώδικα έχουμε ορίσει κανονικές εκφράσεις μέσα σε ορισμένες μεταβλητές για να κάνουμε πιο κατανοητή, κατά την εκτύπωση που θα υπάρξει ως προς την αναγνώριση των κανονικών εκφράσεων, το νόημα της κάθε έκφρασης. Κάτι τέτοιο σημαίνει, πως χρησιμοποιούμε το **DELIMITER** για το τέλος μίας εντολής σε σχέση με το σύμβολο ; . Για το **FLOAT**, το δημιουργούμε για να ορίσουμε πότε μια τιμή είναι πραγματική και αυτό συμβαίνει όταν υπάρχει κόμμα που ορίζει την συνέχεια στην αριθμητική τιμή της καθώς και μέσω του συμβόλου e. Για το **INTCONST** παρατηρούμε πως άμα η τιμή είναι μηδενική ή υπάρχει κάποιος ακέραιος αριθμός τον αναγνωρίζει ως προς την συγκεκριμένη τιμή. Ως προς το **STRING** έχουμε την δυνατότητα να τοποθετούμε κείμενο μέσα στα (‘ “) έτσι ώστε να τα αναγνωρίζει και τα δύο είδη. Κάτι τέτοιο, το ορίσαμε για να έχει παρόμοια χαρακτηριστικά με την γλώσσα C για να είναι πιο εφάμιλλη με αυτή. Για το **IDENTIFIER** ορίζουμε την δυνατότητα να ξεκινάει από γράμμα καθώς και κάτω παύλα τουλάχιστον μια φορά και στην συνέχεια έχει την δυνατότητα να ορίζει κείμενο το οποίο περιέχει αριθμούς ή άλλα γράμματα ή κάτω παύλα αν υπάρχουν άλλα στοιχεία εκτός του πρώτου. Αφετέρου ορίζουμε το **COMMENT**, μέσα στο συγκεκριμένο ορίζουμε τα σχόλια που υπάρχουν στον κώδικα. Τα σχόλια αναγνωρίζονται άμα ξεκινούν από // για σχόλιο της μιας γραμμής ή με το /* για σχολιασμό πάνω σε πολλές γραμμές έχοντας και το αντίστοιχο κλείσιμο που υπάρχει σε σχέση με το σχόλιο τους. Επίσης υπάρχει και ο **OPERATOR** στον οποίο έχουν οριστεί διάφορα σύμβολα που παρατηρούνται μέσα από την γλώσσα Uni-C. Επίσης έχουμε ορίσει να μας αναγνωρίζονται τα κενά και οι κενές γραμμές με την ονομασία **WHITE_SPACES**, όμως θα παρατηρήσουμε και στην συνέχεια πως έχουμε βάλει να μην επιστρέφονται tokens για τα συγκεκριμένα δεδομένα από τον συντακτικό αναλυτή.

```
DELIMITER          ;
FLOAT              [0-9]+\.[0-9]+|[0-9]+\.[0-9]+e[0-9]+
INTCONST           0|[1-9]+[0-9]*
STRING             '.*'|\".*\"
IDENTIFIER         [a-zA-Z_]+([0-9]*|[a-zA-Z_]*)*
COMMENT            \/\*(.|\n)*?\*\/|\/\/.*
OPERATOR           \+|\-|==|!=|\+\+|\*|\|\||\!|\&|\/|\%|=|\*=|\/=|\&\&|\-\-|\<|\>|\<=|\>=|\
+=|\-=
WHITE_SPACES       [ \t]+
```

;

Συνεχίζοντας με τους ορισμούς των κανονικών εκφράσεων, δημιουργήσαμε και την εκτύπωση των τιμών που αφορούν την ανάλυση σε σχέση με το άνοιγμα και το κλείσιμο των καμπυλών, καθώς και των παρενθέσεων. Πιο συγκεκριμένα ορίσαμε ως **OPEN_BRACKETS**, όταν παρατηρούμε να ανοίγει μια αγκύλη. Αντίστοιχα δημιουργήσαμε και την κατάσταση **CLOSE_BRACKETS** όταν παρατηρούμε μια αγκύλη να κλείνει. Παρόμοια βήματα ακολουθήσαμε και στην αναγνώριση των **braces** καθώς και των **parentheses** τα οποία τα αναγνωρίζει κατά το άνοιγμα και το κλείσιμό τους. Όσο αναφορά διάφορες άλλες μεταβλητές όπως το # , την : καθώς και άλλες τιμές τις ορίζουμε μέσα στον **PUNCTUATOR**. Έτσι μπορούμε να κατανοήσουμε κάθε κανονική έκφραση που έχει οριστεί στο πεδίο του δεύτερου τμήματος του ορισμού του λεκτικού αναλυτή μας.

```
OPEN_BRACKETS       \[
CLOSE_BRACKETS      \]
OPEN_BRACES         \{
CLOSE_BRACES        \}
OPEN_PARENTHESES    \(
CLOSE_PARENTHESES   \)
PUNCTUATOR          #|\:|\~|\&|\^|\,
```

Κανόνες Αναγνώρισης

Σε σχέση τώρα με το δεύτερο μέρος που υπάρχει μέσα στο αρχείο περιγραφής του λεκτικού μας αναλυτή, συναντάμε τους κανόνες αναγνώρισης. Αυτό γίνεται αντιληπτό λόγω των συμβόλων % που παρατηρούνται στην αρχή της συγγραφής. Συγκεκριμένα , σε εκείνο το μέρος παρατίθενται οι τιμές που θα επιστραφούν σε περίπτωση που προκύψουν οι συγκεκριμένες κανονικές εκφράσεις, που έχουμε ορίσει. Ιδιαίτερα στις περιπτώσεις που βρούμε κανονική έκφραση που έχει οριστεί ως **DELIMITER**,**FLOAT**,**STRING** ή **INSTCONST** επιστρέφουμε το όνομα που τους έχουμε ορίσει. Όσο αναφορά όμως την περίπτωση του **IDENTIFIER**, υπάρχουν διάφορες εκφράσεις που μας προσδιορίζουν διάφορες τιμές, που έχουν οριστεί ως σταθερές μέσα στο λεκτικό μας αναλυτή. Οι συγκεκριμένες εκφράσεις δεν μπορούν να οριστούν ως ένα απλό κείμενο δηλαδή **IDENTIFIER** από την στιγμή που έχουν διαφορετικές δυνατότητες σε σχέση με τους άλλους **IDENTIFIERS**. Οπότε τις ξεχωρίσαμε και σε περίπτωση που εξεταστούν από τον λεκτικό μας αναλυτή θα εκτυπωθεί στην θέση τους η λέξη **KEYWORD** που υποδηλώνει πως είναι λέξεις κλειδιά. Σε περίπτωση που δεν υπάρξει τέτοια λέξη κατά την εξέταση από τον λεκτικό αναλυτή θα μας τυπωθεί η ορισμένη λέξη **IDENTIFIER**.

```
%%
{DELIMITER}     { return DELIMITER; }
{FLOAT}         { return FLOAT;     }
{STRING}        { return STRING;    }
{INTCONST}      { return INTCONST;  }
{IDENTIFIER}    {
                if      ( !strcmp(yytext,"break"   ) )    return KEYWORD;
                else if ( !strcmp(yytext,"case"    ) )    return KEYWORD;
                else if ( !strcmp(yytext,"func"    ) )    return KEYWORD;
                else if ( !strcmp(yytext,"const"   ) )    return KEYWORD;
                else if ( !strcmp(text,"continue") )    return KEYWORD;
                else if ( !strcmp(yytext,"do"      ) )    return KEYWORD;
                else if ( !strcmp(yytext,"sizeof"  ) )    return KEYWORD;
                else if ( !strcmp(yytext,"struct"  ) )    return KEYWORD;
                else if ( !strcmp(yytext,"switch"  ) )    return KEYWORD;
                else if ( !strcmp(yytext,"void"    ) )    return KEYWORD;
                else if ( !strcmp(yytext,"while"   ) )    return KEYWORD;
                else return IDENTIFIER;
            }
```

Παρατηρούμε όμως πως υπάρχουν και άλλοι κανόνες αναγνώρισης. Συγκεκριμένα, όταν βρεθεί **OPERATOR** θα επιστραφεί στην οθόνη μας η τιμή **OPERATOR**. Όσο αναφορά όμως τα **WHITE_SPACES** καθώς και τα **COMMENT** στην θέση τους δεν εμφανίζεται τίποτα στην οθόνη μας, από την στιγμή που τους έχουμε ορίσει ως τιμή επιστροφής απλός τα σχόλια. Όσο αναφορά τα υπόλοιπα ορίσματα τους έχουμε ορίσει να επιστρέφουν την τιμή που έχουν ονοματιστεί. Δηλαδή στο **OPEN_BRACKETS** επιστρέφουμε το παρόμοιο μήνυμα ονόματος δηλαδή **OPEN_BRACKETS**, αντιστοίχως για όλες τις τιμές που έχουμε ορίσει μέχρι και το **PUNCTUATOR**,θα επιστρέψει και αυτό σε περίπτωση που παρατηρηθεί παρόμοια τιμή του, το όνομα που του έχουμε ορίσει. Σε σχέση τώρα με την **νέα γραμμή**(new line \n) αυτό που θα εκτελεστεί είναι η αύξηση του μετρητή που έχουμε ορίσει στο πεδίο των ορισμών κατά μια μονάδα προς τα πάνω. Αυτό συμβαίνει για να γνωρίζουμε πόσες νέες γραμμές έχουν οριστεί στο εκτελέσιμο αρχείο. Επίσης, για το **EOF** δηλαδή το τέλος του εκτελέσιμου αρχείου επιστρέφεται το κείμενο #END OF FILE# ορίζοντάς μας το τέλος του αρχείου. Τελειώνοντας, μέσω της τελείας τυπώνουμε όλα τα υπόλοιπα που δεν κάνουν match με τίποτα άλλο. Προβάλλοντάς μας μέσω του yyoout το output που έχει το πρόγραμμά μας και μέσω του yytext το κείμενο που σκανάρετε εκείνη την στιγμή από το λεκτικό αναλυτή. Δηλαδή οι δύο τιμές yyoout και yytext θεωρούνται buffers του προγράμματος.

```
{OPERATOR}          { return OPERATOR;}
{WHITE_SPACES}      { /*Do nothing, white space(s)*/}
{COMMENT}           { /*Do nothing, comment detected*/}
{OPEN_BRACKETS}     { return OPEN_BRACKETS; }
{CLOSE_BRACKETS}    { return CLOSE_BRACKETS;}
{OPEN_BRACES}       { return OPEN_BRACES;}
{CLOSE_BRACES}      { return CLOSE_BRACES; }
{OPEN_PARENTHESES}  { return OPEN_PARENTHESES; }
{CLOSE_PARENTHESES} { return CLOSE_PARENTHESES;}
{PUNCTUATOR}        { return PUNCTUATOR;}
\n                  { line++;}
<<EOF>>             { printf("#END-OF-FILE#\n"); exit(0);  }
.                   { fprintf(yyout, "Line=%d, UNKNOWN TOKEN, value=\"%s\"\n", line, yytext);  }
%%
```

Κώδικας Χρήστη

Ξεκινώντας από το τρίτο προαιρετικό μέρος που είναι απαραίτητο για την ορθή εκτέλεση του λεκτικού αναλυτή μας. Το συγκεκριμένο μέρος αποτελεί το πεδίο που έχει οριστεί ο κώδικας του χρήστη. Το παρατηρούμε ότι ξεκινάει και αυτό μέσω των συμβόλων %%. Αρχικά τοποθετούμε μέσα σε ένα πίνακα όλα τα tokens που έχουν οριστεί σε σχέση με τους συγκεκριμένους ορισμούς των κανονικών εκφράσεων. Εν συνεχεία αρχίζει να λειτουργεί το πρόγραμμά μας μέσω της main. Μέσα σε αυτή παρατηρούμε πως έχει οριστεί η ακέραια μεταβλητή token. Το επόμενο που παρατηρείται είναι η χρήση της **IF**. Μέσω της IF παρατηρούμε, αν έχουν οριστεί 3 τιμές μέσω της γραμμής εντολών. Σε περίπτωση που το αρχείο ορίζει τρεις μεταβλητές το πρόγραμμά μας διαβάζει από το αρχείο του δεύτερού μας ορίσματος πληροφορίες τις οποίες τις γράφει μέσα στο τρίτο αρχείο. Σε περίπτωση που τα ορίσματα είναι δύο διαβάζει από το δεύτερο αρχείο πληροφορίες και τις τυπώνει στην οθόνη μας.

```
%%
char *tname[16] = {"DELIMITER","INTCONST","FLOAT", "STRING", "IDENTIFIER" ,
"COMMENT", "WHITE_SPACES", "OPERATOR", "KEYWORD", "OPEN_BRACKETS",
"CLOSE_BRACKETS", "OPEN_BRACES", "CLOSE_BRACES", "OPEN_PARENTHESES",
"CLOSE_PARENTHESES", "PUNCTUATOR"};
int main(int argc, char **argv){
	int token;
if(argc == 3){
		if(!(yyin = fopen(argv[1], "r"))) {
			fprintf(stderr, "Cannot read file: %s\n", argv[1]);
			return 1;
		}
		if(!(yyout = fopen(argv[2], "w"))) {
			fprintf(stderr, "Cannot create file: %s\n", argv[2]);
			return 1;
		}
	}
	else if(argc == 2){
		if(!(yyin = fopen(argv[1], "r"))) {
			fprintf(stderr, "Cannot read file: %s\n", argv[1]);
			return 1;
		}
	}
```

Στην συγκεκριμένη χρήση της επαναληπτικής διαδικασίας **while**, προσπαθούμε να παρατηρήσουμε αν η συνάρτηση yylex έχει διαβάσει χαρακτήρες και από την σύγκριση που υπάρχει μεταξύ τους κάποιοι από τους συγκεκριμένους είναι tokens. Αν αναγνωριστούν tokens είναι αυτά που υπάρχουν μέσα στο συγκεκριμένο αρχείο δηλαδή ανάμεσα στα %%. Σε περίπτωση που υπάρχει κάποιο pattern το οποίο έχει αντιστοιχιστεί με την εντολή return τότε η yylex() συνάρτηση επιστρέφει και αποθηκεύει αυτόματα την τιμή στην μεταβλητή που έχουμε δημιουργήσει, δηλαδή την token.

```
while( (token=yylex()) >= 0){
fprintf(yyout, "Line=%d, token=%s, value=\"%s\"\n", line, tname[token-1], yytext);
	}
	return 0;
}
```

ΠΑΡΑΔΕΙΓΜΑΤΑ ΔΟΚΙΜΩΝ ΚΩΔΙΚΑ

Πρέπει να επισημάνουμε πως ο κώδικάς μας είναι πλήρως λειτουργικός και έχει ότι χρειάζεται για να λειτουργήσει αυτόματα αποδίδοντας όσο αποτελέσματα χρειαστούν για να το αποδείξουμε.

1ο Παράδειγμα

int main () { printf("A long string! 64573*&^$%^!*@&%^;"); }

Line=1, token=KEYWORD, value="int"

Line=1, token=IDENTIFIER, value="main"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=CLOSE_PARENTHESES, value=")"

Line=1, token=OPEN_BRACES, value="{"

Line=1, token=IDENTIFIER, value="printf"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=STRING, value=""A long string! 64573*&^$%^!*@&%^;""

Line=1, token=CLOSE_PARENTHESES, value=")"

Line=1, token=DELIMITER, value=";"

Line=1, token=CLOSE_BRACES, value="}"

Βλέπουμε ότι το πρόγραμμα μας σωστά αναγνώρισε και το άνοιγμα/κλείσιμο τον σημείων στίξης και τα ονόματα, αλλά και την περίπλοκη συμβολοσειρά (string).

2ο Παράδειγμα

3.14e1000

Line=1, token=FLOAT, value="3.14e1000"

Βλέπουμε ότι το πρόγραμμά μας αναγνωρίζει ορθά πολύπλοκους πραγματικούς αριθμούς.

3ο Παράδειγμα

double metavliti = 2.342316

Line=1, token=KEYWORD, value="double"

Line=1, token=IDENTIFIER, value="metavliti"

Line=1, token=OPERATOR, value="="

Line=1, token=FLOAT, value="2.342316"

Το πρόγραμμα μας αναγνώρισε σωστά τα keywords, τους τελεστές αλλά και αγνόησε τα κενά ανάμεσα στις εντολές μας.

4ο Παράδειγμα

if (metavliti == timi $)

Line=1, token=KEYWORD, value="if"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=IDENTIFIER, value="metavliti"

Line=1, token=OPERATOR, value="=="

Line=1, token=IDENTIFIER, value="timi"

Line=1, UNKNOWN TOKEN, value="$"

Line=1, token=CLOSE_PARENTHESES, value=")"

Εδώ βλέπουμε ότι το πρόγραμμά μας σωστά κρίνει ότι ο χαρακτήρας $ είναι λανθασμένος (άγνωστος χαρακτήρας), έχοντας αναγνωρίσει σωστά και τις προηγούμενες εντολές

5ο Παράδειγμα

input.txt

```
int main (int argc*, int argv**) {
    /* The multiline comment
    is this one
    right here */

    // Single Line Comment

    float metavliti = 4;
    if ( metavliti == 12 ) {
        printf("A string");
    }
    return 0;
}
```

```
./uni-c-analyser input.txt
Line=1, token=KEYWORD, value="int"
Line=1, token=IDENTIFIER, value="main"
Line=1, token=OPEN_PARENTHESES, value="("
Line=1, token=KEYWORD, value="int"
Line=1, token=IDENTIFIER, value="argc"
Line=1, token=OPERATOR, value="*"
```

Line=1, token=PUNCTUATOR, value=","

Line=1, token=KEYWORD, value="int"

Line=1, token=IDENTIFIER, value="argv"

Line=1, token=OPERATOR, value="*"

Line=1, token=OPERATOR, value="*"

Line=1, token=CLOSE_PARENTHESES, value=")"

Line=1, token=OPEN_BRACES, value="{"

Line=6, token=KEYWORD, value="float"

Line=6, token=IDENTIFIER, value="metavliti"

Line=6, token=OPERATOR, value="="

Line=6, token=INTCONST, value="4"

Line=6, token=DELIMITER, value=";"

Line=7, token=KEYWORD, value="if"

Line=7, token=OPEN_PARENTHESES, value="("

Line=7, token=IDENTIFIER, value="metavliti"

Line=7, token=OPERATOR, value="=="

Line=7, token=INTCONST, value="12"

Line=7, token=CLOSE_PARENTHESES, value=")"

Line=7, token=OPEN_BRACES, value="{"

Line=8, token=IDENTIFIER, value="printf"

Line=8, token=OPEN_PARENTHESES, value="("

Line=8, token=STRING, value=""A string""

Line=8, token=CLOSE_PARENTHESES, value=")"

Line=8, token=DELIMITER, value=";"

Line=9, token=CLOSE_BRACES, value="}"

Line=10, token=KEYWORD, value="return"

Line=10, token=INTCONST, value="0"

Line=10, token=DELIMITER, value=";"

Line=11, token=CLOSE_BRACES, value="}"

#END-OF-FILE#

Το ολοκληρωμένο πρόγραμμα αυτό σε uni-C δείχνει τις δυνατότητες του αναλυτή μας σε μεγάλο βαθμό, καθώς αποδεικνύει ότι μπορεί να κρίνει σωστά ένα πραγματικό πρόγραμμα.

Βλέπουμε ότι οι γραμμές έχουν μετρηθεί σωστά, οι εντολές έχουν αναγνωριστεί και τα σχόλια μονών και πολλαπλών σειρών έχουν αγνοηθεί.

6ο Παράδειγμα

_metavli1t1_thAt_0134_is_v3333r7_L0ng_0

Line=1, token=IDENTIFIER, value="_metavli1t1_thAt_0134_is_v3333r7_L0ng_0"

Εδώ βλέπουμε με ακραία περίπτωση για τους identifiers. Βλέπουμε ότι το πρόγραμμα το αναγνώρισε και αυτό σωστά.

7ο Παράδειγμα

""

Line=1, token=STRING, value=""""

Εδώ βλέπουμε ότι το πρόγραμμά μας αναγνωρίζει σωστά τις κενές συμβολοσειρές επίσης

8ο Παράδειγμα

#34#
Line=1, token=PUNCTUATOR, value="#"
Line=1, token=INTCONST, value="34"

Line=1, token=PUNCTUATOR, value="#"

Βλέπουμε ότι το πρόγραμμά μας αναγνωρίζει ορθά τα # και επίσης το αριθμό 34 σαν INCONST.

9ο Παράδειγμα

If a<5

Line=3, token=KEYWORD, value="if"
Line=3, token=IDENTIFIER, value="a"
Line=3, token=OPERATOR, value="<"
Line=3, token=INTCONST, value="5"

Βλέπουμε ότι το πρόγραμμα μας σωστά αναγνώρισε την λέξη if σαν KEYWORD και

10ο Παράδειγμα

$sample

Line=4, UNKNOWN TOKEN, value="$"
Line=4, token=IDENTIFIER, value="sample"

Εδώ βλέπουμε ότι το πρόγραμμά μας σωστά κρίνει ότι ο χαρακτήρας $ είναι λανθασμένος (άγνωστος χαρακτήρας), έχοντας αναγνωρίσει σωστά την λέξη sample σαν IDENTIFIER.

11ο Παράδειγμα

Saturation(=[0,1]) specifies how pure a color is; pure red,yellow,green,blue and so on

Line=5, token=IDENTIFIER, value="Saturation"
Line=5, token=OPEN_PARENTHESES, value="("
Line=5, token=OPERATOR, value="="
Line=5, token=OPEN_BRACKETS, value="["
Line=5, token=INTCONST, value="0"
Line=5, token=PUNCTUATOR, value=","
Line=5, token=INTCONST, value="1"
Line=5, token=CLOSE_BRACKETS, value="]"
Line=5, token=CLOSE_PARENTHESES, value=")"
Line=5, token=IDENTIFIER, value="specifies"
Line=5, token=IDENTIFIER, value="how"
Line=5, token=IDENTIFIER, value="pure"

Line=5, token=IDENTIFIER, value="a"
Line=5, token=IDENTIFIER, value="color"
Line=5, token=IDENTIFIER, value="is"
Line=5, token=DELIMITER, value=";"
Line=5, token=IDENTIFIER, value="pure"
Line=5, token=IDENTIFIER, value="red"
Line=5, token=PUNCTUATOR, value=","
Line=5, token=IDENTIFIER, value="yellow"
Line=5, token=PUNCTUATOR, value=","
Line=5, token=IDENTIFIER, value="green"
Line=5, token=PUNCTUATOR, value=","
Line=5, token=IDENTIFIER, value="blue"
Line=5, token=IDENTIFIER, value="and"
Line=5, token=IDENTIFIER, value="so"
Line=5, token=IDENTIFIER, value="on"

Βλέπουμε ότι οι γραμμές έχουν μετρηθεί σωστά, οι εντολές έχουν αναγνωριστεί .

12ο Παράδειγμα

/
Line=6, token=OPERATOR, value="/"

Βλέπουμε ότι το πρόγραμμά μας αναγνωρίζει ορθά τον OPERATOR .

13ο Παράδειγμα

0.121
Line=8, token=FLOAT, value="0.121"

Βλέπουμε ότι το πρόγραμμά μας αναγνωρίζει ορθά πολύπλοκους πραγματικούς αριθμούς.

14ο Παράδειγμα

+.2
Line=9, token=OPERATOR, value="+"
Line=9, UNKNOWN TOKEN, value="."

Line=9, token=INTCONST, value="2"

Εδώ βλέπουμε ότι το πρόγραμμά μας σωστά κρίνει ότι ο χαρακτήρας " . " είναι λανθασμένος (άγνωστος χαρακτήρας), έχοντας αναγνωρίσει σωστά και την προηγούμενη και την επόμενη εντολή.

15ο Παράδειγμα

#ark;limit
Line=10, token=PUNCTUATOR, value="#"
Line=10, token=IDENTIFIER, value="ark"
Line=10, token=DELIMITER, value=";"
Line=10, token=IDENTIFIER, value="limit"

16ο Παράδειγμα

.2e+1
Line=11, UNKNOWN TOKEN, value="."
Line=11, token=INTCONST, value="2"
Line=11, token=IDENTIFIER, value="e"
Line=11, token=OPERATOR, value="+"
Line=11, token=INTCONST, value="1"
"string"
Line=12, token=STRING, value=""string""

Βλέπουμε ότι το πρόγραμμά μας αναγνωρίζει ορθά την συμβολοσειρά.

17ο Παράδειγμα

adb&mvmvmk)9&}[':;
Line=13, token=PUNCTUATOR, value="^"
Line=13, token=IDENTIFIER, value="adb"
Line=13, token=OPERATOR, value="&"
Line=13, token=IDENTIFIER, value="mvmvmk"
Line=13, token=CLOSE_PARENTHESES, value=")"
Line=13, token=INTCONST, value="9"
Line=13, token=OPERATOR, value="&"

Line=13, token=CLOSE_BRACES, value="}"
Line=13, token=OPEN_BRACKETS, value="["
Line=13, UNKNOWN TOKEN, value="'"
Line=13, token=PUNCTUATOR, value=":"
Line=13, token=DELIMITER, value=";"

18ο Παράδειγμα

(foo|bar)*
Line=14, token=OPEN_PARENTHESES, value="("
Line=14, token=IDENTIFIER, value="foo"
Line=14, UNKNOWN TOKEN, value="|"
Line=14, token=IDENTIFIER, value="bar"
Line=14, token=CLOSE_PARENTHESES, value=")"
Line=14, token=OPERATOR, value="*"

Εδώ βλέπουμε ότι το πρόγραμμά μας σωστά κρίνει ότι ο χαρακτήρας | είναι λανθασμένος (άγνωστος χαρακτήρας), έχοντας αναγνωρίσει σωστά την λέξη bar και όλα τα σαν IDENTIFIER.

19ο Παράδειγμα

&&smfk*digit
Line=15, token=OPERATOR, value="&&"
Line=15, token=IDENTIFIER, value="smfk"
Line=15, token=OPERATOR, value="*"
Line=15, token=IDENTIFIER, value="digit"

20ο Παράδειγμα

?image
Line=16, UNKNOWN TOKEN, value="?"
Line=16, token=IDENTIFIER, value="image"

Εδώ βλέπουμε ότι το πρόγραμμά μας σωστά κρίνει ότι ο χαρακτήρας ? είναι λανθασμένος (άγνωστος χαρακτήρας), έχοντας αναγνωρίσει σωστά την λέξη image σαν IDENTIFIER.

21ο Παράδειγμα

for(i=1,i<10,i++)
Line=17, token=KEYWORD, value="for"
Line=17, token=OPEN_PARENTHESES, value="("
Line=17, token=IDENTIFIER, value="i"
Line=17, token=OPERATOR, value="="
Line=17, token=INTCONST, value="1"
Line=17, token=PUNCTUATOR, value=","
Line=17, token=IDENTIFIER, value="i"
Line=17, token=OPERATOR, value="<"
Line=17, token=INTCONST, value="10"
Line=17, token=PUNCTUATOR, value=","
Line=17, token=IDENTIFIER, value="i"
Line=17, token=OPERATOR, value="++"
Line=17, token=CLOSE_PARENTHESES, value=")"

Το ολοκληρωμένο πρόγραμμα αυτό σε uni-C δείχνει τις δυνατότητες του αναλυτή μας σε μεγάλο βαθμό, καθώς αποδεικνύει ότι μπορεί να κρίνει σωστά μια ολόκληρη εντολή. Βλέπουμε ότι οι γραμμές έχουν μετρηθεί σωστά, οι εντολές έχουν αναγνωριστεί .

22ο Παράδειγμα

float * variabl1 = 10*sizeof(*float)

Line=1, token=KEYWORD, value="float"

Line=1, token=OPERATOR, value="*"

Line=1, token=IDENTIFIER, value="variabl1"

Line=1, token=OPERATOR, value="="

Line=1, token=INTCONST, value="10"

Line=1, token=OPERATOR, value="*"

Line=1, token=KEYWORD, value="sizeof"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=OPERATOR, value="*"

Line=1, token=KEYWORD, value="float"

Line=1, token=CLOSE_PARENTHESES, value=")"

Εδώ βλέπουμε ότι ο λεκτικός μας επεξεργαστής αναγνώρισε ορθά και τα keywords μας αλλά τα σημεία στίξης και τους τελεστές, σε μια ιδιαίτερη περίπτωση.

23ο Παράδειγμα

for (i=0; i<=200; i++) printf("Do something");

Line=1, token=KEYWORD, value="for"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=IDENTIFIER, value="i"

Line=1, token=OPERATOR, value="="

Line=1, token=INTCONST, value="0"

Line=1, token=DELIMITER, value=";"

Line=1, token=IDENTIFIER, value="i"

Line=1, token=OPERATOR, value="<="

Line=1, token=INTCONST, value="200"

Line=1, token=DELIMITER, value=";"

Line=1, token=IDENTIFIER, value="i"

Line=1, token=OPERATOR, value="++"

Line=1, token=CLOSE_PARENTHESES, value=")"

Line=1, token=IDENTIFIER, value="printf"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=STRING, value=""Do something""

Line=1, token=CLOSE_PARENTHESES, value=")"

Line=1, token=DELIMITER, value=";"

Εδώ παρατηρούμε ότι το πρόγραμμα μας καταφέρνει να αναγνωρίσει μια δομή for με ευκολία, αναλύοντας σωστά όλους τους τελεστές, τα σημεία στίξης, τα keywords αλλά και τις διάφορες μεταβλητές και ορισμένους τύπους αριθμών, ακόμα και αν η "πυκνότητα" τους είναι αυξημένη.

24ο Παράδειγμα

this_is-test(1)

Line=1, token=IDENTIFIER, value="this_is"

Line=1, token=OPERATOR, value="-"

Line=1, token=IDENTIFIER, value="test"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=INTCONST, value="1"

Line=1, token=CLOSE_PARENTHESES, value=")"

Εδώ βλέπουμε ότι το πρόγραμμά μας έχει αναγνωρίσει σωστά τα IDENTIFIER το (this_is ,test όπως και τα OPEN_ PARENTHESES , CLOSE_ PARENTHESES επίσης το INTCONST (1) και τέλος το OPERATOR (-) .

25ο Παράδειγμα

int main() {

  printf("Hello World!");

```
  return 0;
```

```
}Line=2, token=KEYWORD, value="int"
Line=2, token=IDENTIFIER, value="main"
Line=2, token=OPEN_PARENTHESES, value="("
Line=2, token=CLOSE_PARENTHESES, value=")"
Line=2, token=OPEN_BRACES, value="{"
Line=3, token=IDENTIFIER, value="printf"
Line=3, token=OPEN_PARENTHESES, value="("
Line=3, token=STRING, value=""Hello World!""
Line=3, token=CLOSE_PARENTHESES, value=")"
Line=3, token=DELIMITER, value=";"
Line=4, token=KEYWORD, value="return"
Line=4, token=INTCONST, value="0"
Line=4, token=DELIMITER, value=";"
```

Το ολοκληρωμένο πρόγραμμα αυτό σε uni-C δείχνει τις δυνατότητες του αναλυτή μας σε μεγάλο βαθμό, καθώς αποδεικνύει ότι μπορεί να κρίνει σωστά μια ολόκληρη εντολή. Βλέπουμε ότι οι γραμμές έχουν μετρηθεί σωστά, οι εντολές έχουν αναγνωριστεί .

26ο Παράδειγμα

Fitch Affirms S-T Rating at 'F1+' on Houston, TX GO CP Notes Series G-1

Line=1, token=IDENTIFIER, value="Fitch"

Line=1, token=IDENTIFIER, value="Affirms"

Line=1, token=IDENTIFIER, value="S"

Line=1, token=OPERATOR, value="-"

Line=1, token=IDENTIFIER, value="T"

Line=1, token=IDENTIFIER, value="Rating"

Line=1, token=IDENTIFIER, value="at"

Line=1, token=STRING, value="'F1+'"

Line=1, token=IDENTIFIER, value="on"

Line=1, token=IDENTIFIER, value="Houston"

Line=1, token=PUNCTUATOR, value=","

Line=1, token=IDENTIFIER, value="TX"

Line=1, token=IDENTIFIER, value="GO"

Line=1, token=IDENTIFIER, value="CP"

Line=1, token=IDENTIFIER, value="Notes"

Line=1, token=IDENTIFIER, value="Series"

Line=1, token=IDENTIFIER, value="G"

Line=1, token=OPERATOR, value="-"

Line=1, token=INTCONST, value="1"

Εδώ βλέπουμε ότι έχει διαχωρίσει πολύ σωστά τα IDENTIFIER μεταξύ τους καθώς τα χωρίζει το κενό ,επίσης παρατηρούμε ότι αναγνωρίζει και τα OPERATOR και τα STRING και τα PUNCTUATOR που υπάρχουν .

27ο Παράδειγμα

"digit"

Line=1, token=STRING, value=""digit""

Βλέπουμε ότι το πρόγραμμά μας αναγνωρίζει ορθά την συμβολοσειρά.

28ο Παράδειγμα

+
Line=1, token=OPERATOR, value="+"

Βλέπουμε ότι το πρόγραμμά μας αναγνωρίζει ορθά τον OPERATOR .

29ο Παράδειγμα

design 56@!

Line=1, token=IDENTIFIER, value="design"

Line=1, token=INTCONST, value="56"

Line=1, UNKNOWN TOKEN, value="@"

Line=1, token=OPERATOR, value="!"

Εδώ βλέπουμε ότι το πρόγραμμά μας σωστά κρίνει ότι ο χαρακτήρας " @ " είναι λανθασμένος (άγνωστος χαρακτήρας), έχοντας αναγνωρίσει σωστά και την προηγούμενη και την επόμενη εντολή.

30ο Παράδειγμα

EDIT{Open}-this

Line=3, token=IDENTIFIER, value="EDIT"

Line=3, token=OPEN_BRACES, value="{"

Line=3, token=IDENTIFIER, value="Open"

Line=3, token=CLOSE_BRACES, value="}"

Line=3, token=OPERATOR, value="-"

Line=3, token=IDENTIFIER, value="this"

Εδώ βλέπουμε ότι το πρόγραμμά μας έχει αναγνωρίσει σωστά τα IDENTIFIER το (EDIT ,Open και το this) όπως και τα OPEN_BRACES , CLOSE_BRACES και τέλος το OPERATOR .

31ο Παράδειγμα

^cover<10,drop-a=key

Line=5, token=PUNCTUATOR, value="^"

Line=5, token=IDENTIFIER, value="cover"

Line=5, token=OPERATOR, value="<"

Line=5, token=INTCONST, value="10"

Line=5, token=PUNCTUATOR, value=","

Line=5, token=IDENTIFIER, value="drop"

Line=5, token=OPERATOR, value="-"

Line=5, token=IDENTIFIER, value="a"

Line=5, token=OPERATOR, value="="

Εδώ βλέπουμε ότι το πρόγραμμά μας έχει αναγνωρίσει σωστά τα IDENTIFIER το (cover,drop,a) όπως και τα PUNCTUATOR το ^ , και τέλος τα OPERATOR (< , - , =) .

<u>32ο Παράδειγμα</u>

12/4'asd'

Line=1, token=INTCONST, value="12"

Line=1, token=OPERATOR, value="/"

Line=1, token=INTCONST, value="4"

Line=1, token=STRING, value="'asd'"

#END-OF-FILE#

Παρατηρούμε πως αναγνωρίζει την τιμή 12 και την 4 ως ακέραιες , το / ως Operator και το 'asd' ως string πάντα εκτυπώνεται το END-OF-FILE.

<u>33ο Παράδειγμα</u>

_kala_paei!23

Line=1, token=IDENTIFIER, value="_kala_paei"

Line=1, token=OPERATOR, value="!"

Line=1, token=INTCONST, value="23"

#END-OF-FILE#

Παρατηρούμε πως αναγνωρίζει την τιμή ως _kala_paei Identifier , το θαυμαστικό ως operator και το 23 ως ακέραιο.

<u>34ο Παράδειγμα</u>

paparoupas72%23_kapa

Line=1, token=IDENTIFIER, value="paparoupas72"

Line=1, token=OPERATOR, value="%"

Line=1, token=INTCONST, value="23"

Line=1, token=IDENTIFIER, value="_kapa"

#END-OF-FILE#

Αναγνωρίζει το paparoupas72 ως IDENTIFIER, το % ως OPERATOR , το 23 ως ακέραιο και το _kapa ως IDENTIFIER.

<u>35ο Παράδειγμα</u>

sklave55_hools_all_for_money$$23+32

Line=1, token=IDENTIFIER, value="sklave55_hools_all_for_money"

Line=1, UNKNOWN TOKEN, value="$"

Line=1, UNKNOWN TOKEN, value="$"

Line=1, token=INTCONST, value="23"

Line=1, token=OPERATOR, value="+"

Line=1, token=INTCONST, value="32"

#END-OF-FILE#

Το sklave55_hools_all_for_money το αναγνωρίζει σωστά ως IDENTIFIER , τα δολάρια δεν τα αναγνωρίζει οπότε βάζει την τιμή UNKNOWN TOKEN , το 23 ως INTCONST , το + ως OPERATOR και το 32 και αυτό ως ακέραιο.

36ο Παράδειγμα

m{pouga}tsa me tiri_efaga (stis 9) to vradi^23

Line=1, token=IDENTIFIER, value="m"

Line=1, token=OPEN_BRACES, value="{"

Line=1, token=IDENTIFIER, value="pouga"

Line=1, token=CLOSE_BRACES, value="}"

Line=1, token=IDENTIFIER, value="tsa"

Line=1, token=IDENTIFIER, value="me"

Line=1, token=IDENTIFIER, value="tiri_efaga"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=IDENTIFIER, value="stis"

Line=1, token=INTCONST, value="9"

Line=1, token=CLOSE_PARENTHESES, value=")"

Line=1, token=IDENTIFIER, value="to"

Line=1, token=IDENTIFIER, value="vradi"

Line=1, token=PUNCTUATOR, value="^"

Line=1, token=INTCONST, value="23"

#END-OF-FILE#

Ως IDENTIFIER αναγνωρίζει σωστά το m,pouga,tsa,me,tiri_efaga,stis,to,vradi βρήκε το άνοιγμα της αγκύλης καθώς και το κλείσιμό της , το άνοιγμα της παρένθεσης και το κλείσιμό της , βρήκε τους ακέραιους 9,23 καθώς και το PUNCTUATOR για το ^.

37ο Παράδειγμα

call_55_of_duty*sd(dsa-)=294.12

Line=1, token=IDENTIFIER, value="call_55_of_duty"

Line=1, token=OPERATOR, value="*"

Line=1, token=IDENTIFIER, value="sd"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=IDENTIFIER, value="dsa"

Line=1, token=OPERATOR, value="-"

Line=1, token=CLOSE_PARENTHESES, value=")"

Line=1, token=OPERATOR, value="="

Line=1, token=FLOAT, value="294.12"

Βρήκε ορθά τους identifier , τις παρενθέσεις , τους operator καθώς και την float τιμή.

38ο Παράδειγμα

exw_neura//den tha trelathw

to 3ereis kai esu -23? /*23

56 den me eidane*/

Line=1, token=IDENTIFIER, value="exw_neura"

Line=2, token=IDENTIFIER, value="to"

Line=2, token=INTCONST, value="3"

Line=2, token=IDENTIFIER, value="ereis"

Line=2, token=IDENTIFIER, value="kai"

Line=2, token=IDENTIFIER, value="esu"

Line=2, token=OPERATOR, value="-"

Line=2, token=INTCONST, value="23"

Line=2, UNKNOWN TOKEN, value="?"

#END-OF-FILE#

Βρήκε κανονικά τους identifier , τους ακέραιους , το – ως operator καθώς και το ? ως unknown token. Όμως αυτό που έκανε την μεγαλύτερη διαφορά είναι η αγνόηση των δύο σχολίων καθώς και το enter που υπήρχε ανάμεσα από τι φράσεις.

39ο Παράδειγμα

kobe || jordan || lebron

Line=1, token=IDENTIFIER, value="kobe"

Line=1, token=OPERATOR, value="||"

Line=1, token=IDENTIFIER, value="jordan"

Line=1, token=OPERATOR, value="||"

Line=1, token=IDENTIFIER, value="lebron"

#END-OF-FILE#

Μας προβάλει σωστά και τους IDENTIFIER καθώς και τους OPERATOR.

40ο Παράδειγμα

jordan > kobe = lebron

Line=1, token=IDENTIFIER, value="jordan"

Line=1, token=OPERATOR, value=">"

Line=1, token=IDENTIFIER, value="kobe"

Line=1, token=OPERATOR, value="="

Line=1, token=IDENTIFIER, value="lebron"

#END-OF-FILE#

Μας προβάλει σωστά και τους IDENTIFIER καθώς και τους OPERATOR.

41ο Παράδειγμα

1.messi - 2.Ronaldinho + 3.Ronaldo + 4.Cech * 5.Xavi / 6.Iniesta (7.Kaka [8_Maldini] 9.Rivaldo & 10.Zlatan ^ 11.Zidane % 12.Muller # 13.Giggs @ 14.Henry

Line=1, token=INTCONST, value="1"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="messi"

Line=1, token=OPERATOR, value="-"

Line=1, token=INTCONST, value="2"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Ronaldinho"

Line=1, token=OPERATOR, value="+"

Line=1, token=INTCONST, value="3"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Ronaldo"

Line=1, token=OPERATOR, value="+"

Line=1, token=INTCONST, value="4"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Cech"

Line=1, token=OPERATOR, value="*"

Line=1, token=INTCONST, value="5"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Xavi"

Line=1, token=OPERATOR, value="/"

Line=1, token=INTCONST, value="6"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Iniesta"

Line=1, token=OPEN_PARENTHESES, value="("

Line=1, token=INTCONST, value="7"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Kaka"

Line=1, token=OPEN_BRACKETS, value="["

Line=1, token=INTCONST, value="8"

Line=1, token=IDENTIFIER, value="_Maldini"

Line=1, token=CLOSE_BRACKETS, value="]"

Line=1, token=INTCONST, value="9"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Rivaldo"

Line=1, token=OPERATOR, value="&"

Line=1, token=INTCONST, value="10"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Zlatan"

Line=1, token=PUNCTUATOR, value="^"

Line=1, token=INTCONST, value="11"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Zidane"

Line=1, token=OPERATOR, value="%"

Line=1, token=INTCONST, value="12"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Muller"

Line=1, token=PUNCTUATOR, value="#"

Line=1, token=INTCONST, value="13"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Giggs"

Line=1, UNKNOWN TOKEN, value="@"

Line=1, token=INTCONST, value="14"

Line=1, UNKNOWN TOKEN, value="."

Line=1, token=IDENTIFIER, value="Henry"

#END-OF-FILE#

Μας προβάλλει σωστά τους ακέραιους αριθμούς καθώς και τις τιμές όλων των identifier τους αναγνωρίζει σωστά, βρίσκει επίσης ορθά όλους τους operator που χρησιμοποιήθηκαν καθώς και τις παρενθέσεις που υπάρχουν ανάμεσα. Βρίσκει όλους τους PUNCTUATORS καθώς και την σωστή μη αναγνώριση από το @ καθώς και την τελεία ως UNKNOWN TOKEN.

42ο Παράδειγμα

47^ *re

Line=1, token=INTCONST, value="47"

Line=1, token=PUNCTUATOR, value="^"

Line=1, token=OPERATOR, value="*"

Line=1, token=IDENTIFIER, value="re"

Βλέπουμε ότι το πρόγραμμα στο παράδειγμα 47^ *re έχει αναγνωρίσει σωστά όλα τα στοιχεία και έχει αγνοήσει τον κενό χαρακτήρα.

43ο Παράδειγμα

.:{r4l\

Line=2, UNKNOWN TOKEN, value="."

Line=2, token=PUNCTUATOR, value=":"

Line=2, token=OPEN_BRACES, value="{"

Line=2, token=IDENTIFIER, value="r4l"

Line=2, UNKNOWN TOKEN, value="\"

Στο παραπάνω παράδειγμα .:{r4l\ το πρόγραμμα μας έχει εντοπίσει τους λανθασμένους χαρακτήρες " . " και " \ " και τους υπόλοιπους τους έχει αντιστοιχήσει στις σωστές κατηγορίες.

44ο Παράδειγμα

command ..pr2%

Line=3, token=IDENTIFIER, value="command"

Line=3, UNKNOWN TOKEN, value="."

Line=3, UNKNOWN TOKEN, value="."

Line=3, token=IDENTIFIER, value="pr2"

Line=3, token=OPERATOR, value="%"

Στο παράδειγμα μας command ..pr2% το πρόγραμμα διαβάζει και αγνοεί τον κενό χαρακτήρα και καταλαβαίνει τους αγνώστους χαρακτήρες ¨ . ", τα υπόλοιπα στοιχεία έχουν κατανεμηθεί στις σωστές κατηγορίες.

45ο Παράδειγμα

+ =(34^[[1;2Afr

Line=4, token=OPERATOR, value="+"

Line=4, token=OPERATOR, value="="

Line=4, token=OPEN_PARENTHESES, value="("

Line=4, token=INTCONST, value="34"

Line=4, UNKNOWN TOKEN, value="

Line=4, token=OPEN_BRACKETS, value="["

Line=4, token=INTCONST, value="1"

Line=4, token=DELIMITER, value=";"

Line=4, token=INTCONST, value="2"

Line=4, token=IDENTIFIER, value="Afr"

Στο παραπάνω παράδειγμα + =(34^[[1;2Afr ο κώδικας μας έχει αναγνωρίσει τις παρενθέσεις τις αγκύλες και τους αγνώστους χαρακτήρες ενώ τα υπόλοιπα στοιχεία έχουν κατανεμηθεί στις σωστές κατηγορίες.

46ο Παράδειγμα

}team 15.15team,

Line=5, token=CLOSE_BRACES, value="}"

Line=5, token=IDENTIFIER, value="team"

Line=5, token=FLOAT, value="15.15"

Line=5, token=IDENTIFIER, value="team"

Line=5, token=PUNCTUATOR, value=","

Με παράδειγμα το }team 15.15team, ο κώδικας εντοπίζει την αγκύλη και την κατηγοριοποιεί, όπως και τα στοιχεία του παραδείγματος σε σωστές κατηγορίες.

47ο Παράδειγμα

><ergasia meros a3" april

Line=6, token=OPERATOR, value=">"

Line=6, token=OPERATOR, value="<"

Line=6, token=IDENTIFIER, value="ergasia"

Line=6, token=IDENTIFIER, value="meros"

Line=6, token=IDENTIFIER, value="a3"

Line=6, UNKNOWN TOKEN, value="""

Line=6, token=IDENTIFIER, value="april"

Με παράδειγμα ένα string ><ergasia meros a3" april το πρόγραμμα αναγνωρίζει τα άγνωστα στοιχεία καθώς και τα operator τα οποία και κατηγοριοποιεί.

48ο Παράδειγμα

ifelse

Line=1, token=IDENTIFIER, value="ifelse"

Με παράδειγμα την λέξη ifelse το πρόγραμμα καταλαβαίνει πως δεν είναι keyword και τα βάζει σε σωστή κατηγορία.

49ο Παράδειγμα

else if

Line=2, token=KEYWORD, value="else"

Line=2, token=KEYWORD, value="if"

50ο Παράδειγμα

else

Line=3, token=KEYWORD, value="else"

51ο Παράδειγμα

if

Line=4, token=KEYWORD, value="if"

52ο Παράδειγμα

for i

Line=5, token=KEYWORD, value="for"

Line=5, token=IDENTIFIER, value="i"

Ενώ στα παραπάνω 4 παραδείγματα καταλαβαίνει πως οι λέξεις που του δίνουμε είναι keyword και τα κατηγοριοποιεί στη κατηγορία αυτή.

53ο Παράδειγμα

printf cout integer"23"

Line=6, token=IDENTIFIER, value="printf"

Line=6, token=IDENTIFIER, value="cout"

Line=6, token=IDENTIFIER, value="integer"

Line=6, token=STRING, value=""23""

Στη δοκιμή με παράδειγμα printf cout integer"23" παρατηρούμε ότι το πρόγραμμα καταλαβαίνει ότι ό,τι είναι μέσα σε εισαγωγικά είναι string και το βάζει στη σωστή κατηγορία.

54ο Παράδειγμα

print("ok")

Line=7, token=IDENTIFIER, value="print"

Line=7, token=OPEN_PARENTHESES, value="("

Line=7, token=STRING, value=""ok""

Line=7, token=CLOSE_PARENTHESES, value=")"

55ο Παράδειγμα

float *a=(varable not integer);

Line=8, token=KEYWORD, value="float"

Line=8, token=OPERATOR, value="*"

Line=8, token=IDENTIFIER, value="a"

Line=8, token=OPERATOR, value="="

Line=8, token=OPEN_PARENTHESES, value="("

Line=8, token=IDENTIFIER, value="varable"

Line=8, token=IDENTIFIER, value="not"

Line=8, token=IDENTIFIER, value="integer"

Line=8, token=CLOSE_PARENTHESES, value=")"

Line=8, token=DELIMITER, value=";"

Στο παράδειγμα float *a=(varable not integer); το πρόγραμμα καταλαβαίνει τις εντολές keyword καθώς και παρενθέσεις επίσης βλέπουμε ότι καταλαβαίνει και αφαιρεί τα τα κενά στοιχεία.

56ο Παράδειγμα

int *b=(pointer@) 3+45; /* work */

Line=9, token=KEYWORD, value="int"

Line=9, token=OPERATOR, value="*"

Line=9, token=IDENTIFIER, value="b"

Line=9, token=OPERATOR, value="="

Line=9, token=OPEN_PARENTHESES, value="("

Line=9, token=IDENTIFIER, value="pointer"

Line=9, UNKNOWN TOKEN, value="@"

Line=9, token=CLOSE_PARENTHESES, value=")"

Line=9, token=INTCONST, value="3"

Line=9, token=OPERATOR, value="+"

Line=9, token=INTCONST, value="45"

Line=9, token=DELIMITER, value=";"

Το πρόγραμμα με παράδειγμα int *b=(pointer@) 3+45; /* work */ το οποίο περιέχει σχόλιο παρατηρούμε ότι το αφαιρεί, ενώ καταλαβαίνει τους αγνώστους χαρακτήρες καθώς και όλα τα υπόλοιπα στοιχεία τα βάζει σε σωστές κατηγορίες.

57ο Παράδειγμα

#include<stdio>//libr

Line=5, token=PUNCTUATOR, value="#"

Line=5, token=IDENTIFIER, value="include"

Line=5, token=OPERATOR, value="<"

Line=5, token=IDENTIFIER, value="stdio"

Line=5, token=OPERATOR, value=">"

Ευχαριστίες,

Σας ευχαριστούμε πολύ για τον χρόνο σας