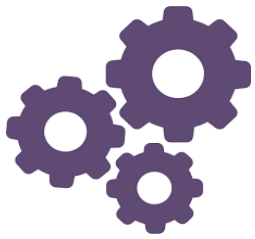
Εργασία Μεταγλωττιστών

Τμήμα Α2

Ομάδα 15

27/4/2021



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 Β1: Σπυρίδων Φλώρος

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

Κατάλογος περιεχομένων

Εισαγωγή

- Με τι θα ασχοληθούμε παρακατω:

  Σε αυτό το εργαστηριακό κομμάτι θα ασχοληθούμε με μια γεννήτρια συντακτικών αναλυτών για την γλώσσα C και C++ που ονομαζεται BISON.

- Τι είναι το BISON;

Το BISON όπως προείπαμε ειναι μια γεννήτρια συντακτικών αναλυτών που μετατρέπει την γραμματική μιας contax-free γραμματικής σε ένα LALR (Look-Ahead Left-to-right parse, Rightmostderivation). Το BISON κατασκευάστηκε ως μια βελτιωμένη έκδοση του YACC, ότι συναταμε στο ενα εργαλείο συναντάμε και στο άλλο, είναι δύσκολο να βρεθεί εντολή ή

σύνταξη στο ένα η οποία δεν θα υπάρχει στο άλλο.

- Σε ποια γλώσσα βασίζεται:

Το εργαλείο BISON βασίζεται και συντάσσεται στη γλώσσα C. Για αυτό και όλος του ο κωδικας διατηρεί πολλά στοιχεία από την ίδια την γλώσαα. Για την αναγνωριση των λεκτικών μονάδων χρησιμοποιεί κανονικές εκφράσεις.

- Πως δημιουργείται ο λεκτικός αναλυτής:

Αρχικά έχουμε ένα πηγαίο κώδικα με κατάληξη " .y ". Αυτο το αρχείο το μεταγλωτίζουμε μέσω του BISON και μας παραγει ενα αρχείο με κατάληξη " .c " το οποίο μπορούμε να μεταγλωτίσσουμε με τον μεταγλωτιστή της γλώσσας που μας παράγει ενα κατάλληλα εκτελεσιμο αρχείο, στο δίνοντας του μια πράξη ή μια έκραση ή και ένα αρχείο, θα μας εμφανισει τα κατάλληλα αποτελέσματα.

Σελίδα στο GITHUB

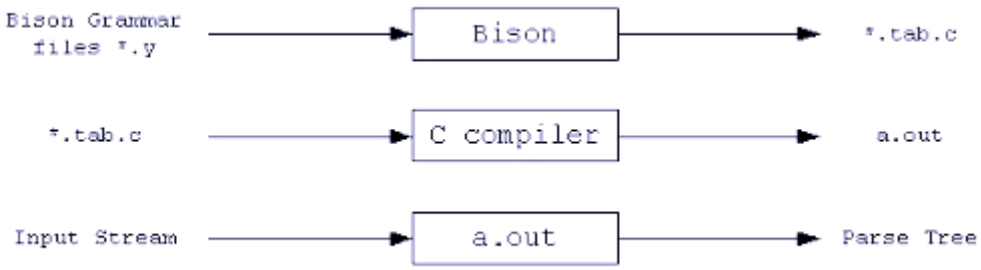
Έχουμε δημιουργήσει μια σελίδα που βρίσκεται στο Github, μέσω εκείνης μπορούμε να έχουμε όλοι μας την δυνατότητα να μοιράσουμε την πληροφορία και την πορεία που είχαμε πάνω στην εργασία μας. Μπορείτε να το επισκεφτείτε το παρακάτω link:

https://github.com/dnnis/uni-C-Analyser

- Τι θα δουμε σε αυτη την εργασία

Το κυρίως θέμα σε αυτό το PDF είναι η δημιουργία λεκτικού αναλυτή BISON καθώς και η λειτουργια του. Παρακάτω παραθέτουμε επεξηγηματική ανάλυση για το πως δημηουργείταιτρέχει και παράγει αποτέλεσμα ένας αναλυτής BISON. Επιπλέον υπάρχουν σχόλια για κάθε κομμάτι κώδηκα καθώς και σχήμα λειτουργίας του αναλυτή. Τέλος υπαρχουν παραδείγματα λειτουργίας για όλες τις πτυχές που καλύπτει ο κώδηκας.

Διαγραμματική λειτουργία του λεκτικού αναλυτή BISON:



Στο παραπάνω σχήμα διακρίνεται η ακριβής λειτουργεία του αρχείου. Διακρίνεται αρχικά ενα bison parser αρχειο με κατάληξη .y επειτα μεσω ενώς μεταγλωτιστή ένα αρχείο .c και στο τέλος παράγεται ενα αρχείο a.out το οποίο είναι το εκτελέσιμο αρχείο. Αλλά ας το δούμε αναλυτικά .Έχουμε ενα πηγαίο αρχείο το οποιο είναι bison parser δηλαδή ενα έτοιμο αρχείο

γραμμένο σε bison γραμματική, με κατάληξη " .y ". Μέσω ενώς bison μεταγλωτιστή το αρχείο μεταγλωτίζεται σε parser αρχείο με κατάληξη " .c " το οποίο μπορει να μεταγλωτιστεί πλεον απο τον μεταγλωτιστή της γλώσσας C (gcc), όπως και κανει στο επόμενο βήμα. Αυτό έχει ως αποτέλεσμα την δημιουργεία ενώς εκτελέσιμου αρχείου .out. Το οποίο δίνοντας του δεδομένα μας επιστρέφει τα αποτελέσματα που έχουν ορισθεί στον κώδηκα.

Δομή εκτελέσιμου κώδηκα πρώτου υποερωτήματος.

Η δομή του προγράματος με κατάληξη " .y " έχει ολα τα δομικά χαρακτηρηστικά που περιέχει ένα parser αρχείο σε γραμματική bison. Δηλαδή το πρώτο κομμάτι που παρατηρείται ειναι το εισαγωσικό τμήμα ξεκινάει με τα εξής σύμβολα %{ και τελειωνει με τα εξής %}, οτι παραθέτεται μέσα σε αυτά είναι το εισαγωγικό τμήμα του κώδημα. Αμέσως μετά ξεκινάνε οι δηλώσεις bison, σε αυτό το κομμάτι υπάρχουν όλοι οι τύποι μεταβλητών που θα χρησημοποιηθούν στον κώδηκα. Παρακάτω βρήσκεται το τμήμα κανόνων. Είναι οι συθήκες, οι εντολές και οι πράξεις που ορίζονται απο τον χρήστη για την διαμόρφωση του προγράμματος στις απαιτήσεις του εκάστοτε χρήστη. Αυτό το τμήμα βρήσκεται αναμεσα απο τα εξής σύμβολα " %% ". Δηλαδή σε κάθε πρόγραμμα bison η αρχή του τμήματος κανόνων ξεκινάει και τελειώνει με αυτό το σύμβολο. Στο τελευταίο τμήμα βρήσκεται το κυρίως πρόγραμμα το οποίο ειναι γραμμένο σε γλώσσα C ή C++. Είναι το main τμημα του

προγράμματος. Παρακάτω παραθέτεται ένα πρότυπο σχήμα πριν την περαιτέρω ανάλυση τους.

%{

Εισαγωγικό Τμήμα

%}

Δηλώσεις bison

%%

Περιγραφή γραμματικής

%%

Κυρίως πρόγραμμα C ή C++

Στο εισαγωγικό τμήμα του προγράμματος βρίσκοντα συνήθως όλες οι βιβλιοθήκες που ειναι απαραίτητες για την περαίωση του προγραμματος καθώς και τυχών συναρτήσεις. Οτι εμπεριέχεται σε αυτό το τμήμα περνάει ατόφιο στο παραγόμενο αρχείο " .c " με τον τρόπο που προείπαμε.

Στο επόμενο μέρος << Δηλώσεις bison>> αναγράφονται όλα τα σύμβολα της γραμματικής καθώς και δηλώσεις αρχικών συμβόλων, τερματικών συμβόλων, μη τερματικών συμβόλων, οι

προταιρεότητες καθώς και κάποιες παράμετροι που επηρεάζουν τον συντακτικό αναλυτή.

Στο τμήμα περιγραφή γραμματικής υπάρχουν εκφράσεις γραμμένες σύμφωνα με τους γραμματιακούς κανόνες. Εδώ μέσα δηλώνονται το τι θα πραγματοποιήσει ο κώδηκας.

Τέλος υπάρχει το τμήμα του κυρίως προγράμματος,εδώ γράφεται ο main κώδηκας, περιέχονται συναρησεις που χρησιοποιούνται απο τον συντακτίκο αναλυτή. Ότι υπαρχει σε αυτό το τμήμα μεταφέρεται στο παραγόμενο " .c " αρχειο.

Σχολιασμός Κώδικα “simple-bison-code.y”

Τμήμα ορισμού Α'(εισαγωγικό τμήμα)

Ξεκινώντας την ανάλυση του πρώτου υποερωτήματος της εργαστηριακής άσκησης Β1 παίρνουμε το αρχείο " simple-bison-code.y " το οποίο ειναι τροποποιημένο ώστε να εκτελεί το ζητούμενο, δηλαδή έχουν προστεθεί κομμάτια κώδηκα bison ώστε το πρόγραμμα να αναγνωρίζει την πρόσθεση, την αφαίρεση, τον πολλαπλασιασμό και την διαίρεση ακεραίων αριθμών του δεκαδικού συστήματος αλλά και μεταβλητών. Επιπλέον δημιουργήθηκε ενα κομμάτι κώδηκα ώστε τα αποτελέσματα να εμφανίζονται στην οθόνη. Στο εισαγωγικό του τμήμα βλέπουμε τις εξής δηλώσεις:

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

int yylex(void);

void yyerror(char *);

υπάρχουν δηλωμένες δύο βιβλιοθήκες και δύο συναρτήσεις, οι yylex και yyerror. Η yylex παραγεται απο το parser αρχείο και η δουλεια της είναι να κατασκευάζει έναν συντακτικό αναλυτή ενώ η yyerror είναι υπεύθυνη για την εκτύπωση μήνυματος λάθους στην οθόνη.

Τμήμα ορισμου Β'(δηλώσεις BISON)

Στο τμήμα δηλώσεων bison έχουμε τις εξής δηλώσεις

%left POW

%left PLUS MINUS

%left DIV MULTI

%right EQ

και οι τρεις δηλώσεις δηλώνουν την προταιρεότητα των πραξεων των μεταβλητών.

Τμήμα ορισμού Γ'(περιγραφή γραμματικής)

Στο συγκεκριμένο τμήμα αναγράφονται τα ορίσματα των συντακτικών κανόνων. Με την είσοδο χαρακτήρων ή προγραματων ή μεταβλητών αντιστοιχίζεται στην κατάλληλη κατηγορία και εκτελείται ο κώδηκας δεξιά του.program:

program expression NEWLINE { printf("\t* Result: %d\n", $2); }

|

```
;

expression:

        INTCONST                        { $$ = $1; }

       | VARIABLE                       { $$ = $1; }

       | VARIABLE VARIABLE                      { $$ = $1; }

       | expression PLUS expression             { $$ = $1 + $3;}

       | expression MINUS expression             { $$ = $1 - $3;}

       | expression MULTI expression             { $$ = $1 * $3;}

       | expression DIV expression              { $$ = $1 / $3;}

       | expression POW expression              { $$ = pow($1,$3); }

       | expression expression EQ expression SEMI { $$ = $1 = $3;}

       | expression EQ expression SEMI          { $$ = $1 = $3;}

       ;

expr: expression '\n'



expressions: expressions expr

  | expr

  ;
```

Συγκεκριμένα όταν ο συντακτικός αναλυτής πάρει σαν είσοδο μια εκφραση και ενα enter τότε μπαίνει στην κατηγορία **program expression NEWLINE** και μας δίνει το αποτέλεσμα στην οθόνη. Περνόντας στην κατηγορία **expression** υπάρχουν πολλές υποκατηγορίες όπως **INTCONST** που μπαινει οταν υπάρχει σαν είσοδος ενας αριθμός τον οποίο και αρχικοποιεί στην μεταβλητη i ενώ εκτυπώνει intconst,**VARIABLE** που μπαίνει όταν έχει σαν είσοδο μία μεταβλητή την οποία και αρχικοποιεί στην μεταβλητή i, **VARIABLE VARIABLE** που μπαινει οταν υπάρχουν δύο μεταβλητές σαν ορισμα, **expression PLUS expression** μπαίνει οταν υπάρχει αριθμητική πράξη της πρόσθεσης, **expression MINUS expression** μπαίνει οταν υπάρχει σαν όρισμα η αριθμητική πράξη της αφαίρεσης, **expression MULTI expression** μπαινει οταν υπάρχει σαν είσοδος η αριθμητική πράξη του πολλαπλασιασμού, **expression DIV expression** μπαίνει όταν υπάρχει σαν είσοδος η πράξη της διαίρεσης, **expression POW expression** μπαίνει όταν υπάρχει σαν είσοδοσ αριθμητική πράξη της υψωσης σε δύναμη, **expression expression EQ expression SEMI** μπαίνει όταν οι δύο πρώτες μεταβλητές παίρνουν την τιμή της τρίτης μεταβλητής και τελειωνει με ερωτηματικό και η **expression EQ expression SEMI** που μπαίνει οταν όλες οι μεταβλητές ειναι μεταξύ τους ίσες και τελειώνει με ερωτηματικό. Στην επόμενη κατηγορία **expr: expression '\n'** ορίζει ενα expression ακολουθειτε παντα απο αλλγή γραμής για να είναι expression.

Τμήμα ορισμού Δ'(περιγραφή γραμματικής)

Σε αυτο το τμημα ορισμού θα αναφερθούμε σε ότι βρήσκεται ανάμεσα απο τα εξής σύμβολα

%%. Στον παρακάτω κώδηκα:

```
int yylex() {
char buf[100];
char num = 0;
int zero = 0;
char c;
c = getchar();
// Ομάδα 15: Για το συγκεκριμένο υποερώτημα (για απλές αριθμητικές πράξες), αγνοούνται οι χαρακτήρες "{","}","(",")","[","]" και \n
while (c == ' ' || c == '\t' || c == '\n' || c == '(' || c == ')' || c == '[' || c ==']' || c == '{' || c == '}' )
{
  yylval = 0;
  c = getchar();
}
if ((c >= 'A' && c <= 'Z') ||
  (c >= 'a' && c <= 'z') ||
  (c == '_'))
{
```

```
    sprintf(buf, "%c", c);
    c = getchar();
    while((c >= 'A' && c <= 'Z') ||
        (c >= 'a' && c <= 'z') ||
        (c >= '0' && c <= '9') ||
        (c == '_'))
    {
       sprintf(buf, "%s%c", buf, c);
       c = getchar();
    }
    ungetc(c, stdin);
    yylval = 0;
    printf("\tScanner returned: VARIABLE (%s)\n", buf);
    return VARIABLE;
  }
while (c >= '0' && c <= '9')
  {
    if (zero > 0)
```

```
    {
        zero = 0;
        yyerror("integer starting with zero");
        exit(1);
    }
    // Ομάδα 15: Bugfix: "&& num == 0" διότι ο αρχικός κώδικα δεν δεχόταν αριθμούς όπως
100,101,342301 (που περιείχαν γενικά "0")
    if (c == '0' && num == 0) zero++;
    if (num == 0) yylval = 0;
    yylval = (yylval * 10) + (c - '0');
    num = 1;
    c = getchar();
  }
```

έχουμε την yylex η οποία όπως αναφέρθηκε παραπάνω δημιουργεί εναν συντακτικό αναλυτή αυτόνομο. Φτιάχνει ενα πινακα buf εκατό θεσεων και αρχικοποιεί την num ίση με το μηδεν καθώς και την zero επίσης και ζητάει έναν χαρακτήρα απο το πληκτρολόγειο. Αν ο χαρακτήρας ειναι κενός ή tab τον αγνωεί και ζηταεί άλλον. Με επέμαση της ομάδας 15 για το συγκεκριμένο υποερώτημα αναπτύθχηκε μια while η οποία αγνοεί τους εξής χαρακτήρες "{","}","(",")","[","]" και το '' \n ''. Επειτα εκτελούνται δύο συνθήκες if η πρώτη ανιχνεύει αν ο

χαρακτήρας ειναι μεταβλητή και η δευτερη αν ο επόμενος χαρακτήρας μετά τον πρώτο ειναι ψηφίο, έπειτα το εκτυπώνει στον buffer με την εντολή sprintf και ζητάει πάλι επόμενο χαρακτήρα, τον ελεγχει και γυρίζει variable αν αυτο ικανοποιεί την συνήκη.

Στον επόμενο βρόχο while ελέγχει κάθε χαρακτήρα που είναι αριθμός και τον βαζει στην συνάρτηση yylval. Στην συνθήκη if έχει γίνει διόρθωση απο την ομάδα 15 διότι το πρόγραμμα δεν ανταποκρινόταν οταν έβρησκε τον αριθμό μηδέν. Στο παρακάτω if

```
if (num) {
    ungetc(c, stdin);
    printf("\tScanner returned: INTCONST (%d)\n", yylval);
    return INTCONST; }
```

ελέγχει αν το num ειναι αριθμός και το γυρνάει.

Στο παρακάτω κομμάτι κώδηκα:

```
else if (c == '-')
{
    printf("\tScanner returned: MINUS (%c)\n", c);
    return MINUS;
}
else if (c == '+')
{
```

```
    printf("\tScanner returned: PLUS (%c)\n", c);
     return PLUS;
  }
  else if (c == '\n')
  {
    yylval = 0;
    printf("\tScanner returned: NEWLINE (\\n)\n");
    return NEWLINE;
  }
  else if (c == EOF)
  {  printf("\tScanner returned: EOF (EOF)\n");
    exit(0);  }
```

ελέγχει αν υπάρχουν σύμβολα αριθμητικών πράξεων " + ", " -" καθώς "\n" kai eof και επιστέφει το ανάλογο αποτέλεσμα. Παρακάτω το επόμενο κομμάτι κώδηκα αναγνωρήζει αν υπάρχουν τα σύμβολα " / ", " * ", " =" και " ; " ενώ για οποιοδήποτε άλλο συβμολο καλεί την yyerror και εμφανίζει μήνυμα λάθους " invalid character".

```
 else if (c == '/')
  {
    printf("\tScanner returned: DIVISION (DIV)\n");
    return DIV;
```

```
  }
else if (c == '*')
  {
     printf("\tScanner returned: MULTIPLY (MULTI)\n");
     return MULTI;
  }
 else if (c == '=')
  {
     printf("\tScanner returned: EQUALS (EQ)\n");
     return EQ;
  }
else if (c == ';')
  {
     printf("\tScanner returned: SEMICOLON (SEMI)\n");
     return SEMI;
  }
 yyerror("invalid character");
}
```

Τμήμα ορισμού Ε'(κυρίως πρόγραμμα)

Τέλος το τελευταίο κομματι ειναι το κυρίως πρόγραμμα, σε αυτή τη περίπτωση το main

τμημα καλεί την συνάρτηση yyparse η οποία ξεκινάει τον συντακτικό αναλυτή.

```
int main(void) {
 yyparse();
 return 0;
}
```

Δομή εκτελέσιμου κώδηκα δεύτερου υποερωτήματος.

Στο δεύτερο υποερώτημα της εργαστηριακής άσκησης Β1 τροποποιηθηκε το πρόγραμμα καταλλήλως απο το αρχείο simple-bison-code.y ώστε να αναγνωρίζει περισσότερες εκφράσεις. Το αρχείο ονομάστηκε bison-SA.y και παρακάτω υπάρχει αναλυτική επεξήγηση του κώδηκα.

Σχολιαμός κώδηκα " bison-SA.y "

Η μορφολογία του κώδηκα ακολουθεί αυτή που έχει αναλυθεί στο κεφάλαιο ΔΟΜΗ ΕΚΤΕΛΕΣΙΜΟΥ ΚΩΔΗΚΑ. Θα υπάρχουν με την σειρά οι κατηγορίες: εισαγωγικό τμήμα, δηλώσεις bison, περιγραφή γραμματικής και κυρίως πρόγραμμα.

Τμημα ορισμού Α'(εισαγωγικό τμήμα)

Στο παρακάτω τμημα του κώδηκα περικλείεται απο το σύμβολο αρχης %{ και το τερματικο συμβολο %}. Εδώ ειναι δηλωμένες οι βιβλιοθήκες και οι συναρτήσεις που εχουν κληθεί

παρακάτω, επίσης σε αυτό το κομμάτι δηλώνονται αρχικοποιήσεις μεταβλητών, αρχεία καθώς και δηλώσεις define. Η συνάρτηση yylex όπως και στο προηγούμενο κώδηκα ειναι η συνάρτηση που κατασκευάζει τον συντακτικό αναλυτή. Ενώ η yyerror είναι η συνάρτηση που καλείται όταν πρέπει να εμφανιστεί μήνυμα λαθους.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int yylex(void);
void yyerror(char *);
```

Τμήμα ορισμού Β'(δηλώσεις bison)

Στο δεύτερο τμήμα του κωδηκα υπάρχουν οι δηλωμένες προταιρεοτητες του κώδηκα και ειναι οι:

```
%left POW
%left PLUS MINUS
%left DIV MULTI
%right EQ
```

Τα left – right είναι αυτα που καθορίζουν την προταιρεοτητα της πράξης. Αν υπαρχε %left σημαινει οτι η προταιρεοτητα ξεκινάει απο τα αριστερά ενώ αν υπάρχει %right η

προταιρεοτητα ξεκινάει απο τα δεξιά προς τα αριστερά.

Τμήμα ορισμού Γ'(περιγραφή γραμματικής)

Στο τμήμα αυτό δημιουργούνται οι κανόνες της γραμματικής. Σύμφωνα με αυτό το τμήμα το πρόγραμμα καταλαβαίνει και κατηγοριοποιεί την είσοδο που το δίνουμε . Είναι το κομμάτι κώδηκα που στην αρχή του και στο τέλος του υπάρχουν τα εξής σύμβολα " %% ". παρακάτω παραθέτονται οι συνθήκες του προγράμματος.

```
expr_part:
        INTCONST
      | ID
      ;
```

στο παραπάνω κομμάτι κώδηκα ορίζεται το τι μπορεί να είναι κομμάτι μιασ εκφρασης, αν είναι δηλαδή ένας χαρακτήρας ή ενας αριθμός. Παρακάτω παραθέτεται το κομμάτι κώδηκα ποθ ορίζει ποιες είναι οι εκφράσεις υπο επεξεργασία. Στις εκφράσεις έχουν προστεθεί οι αρηθμητικές πράξεις της πρόσθεσης, της αφαίρεσης, του πολλαπλασιασμού και της διαίρεσης καθώς και η ισοδυναμία.

```
expr_proc:
        expr_part expr_part EQ expr_part { $$ = $1 = $3; }
      | expr_part PLUS expr_part        { $$ = $1 + $3; }
      | expr_part MINUS expr_part       { $$ = $1 - $3; }
```

```
| expr_part MULTI expr_part       { $$ = $1 * $3; }
| expr_part DIV expr_part         { $$ = $1 / $3; }
| expr_part EQ_expr_part          { $$ = $1 = $3; }
| expr_part POW expr_part         { $$ = pow($1,$3); }
;
```

Συνεχίζοντας υπάρχουν οι κατηγοριες **body:** στην οποία ορίζεται το σώμα του κώδηκα, η **in_bra:** στην οποία οριζεται τι βρήσκεται μετα στις αγκυλες, η **arguments:** στην οποια ορίζεται τι θα είναι όρισμα της συνάρτησης, η **func_par:** η οποία ορίχει τι θεωρείται όρισμα της συνάρτησης, η **declaration:** που ειναι ο ορισμός μιας μςταβλητης, η **assignment:** που ορίζει τι θεωρείται αναθεση σε μεταβλητή και η **valid:** που ορίζει τι ειναι συντακτικά σωστό.

Τμήμα ορισμού Δ'(περιγραφή γραμματικής)

Στη συνέχεια του κώδηκα βλέπουμε την yylex αυτή η συνάρτηση οπως προείπαμε δημιουργεί εναν συντακτικό αναλυτή, με καποιους ελέγχους. Οι έλεγχοι αυτοι είναι για το αν βρήσκεται μεσα σε αγκύλες ή μεσα σε παρενθέσεις. Αν ισχύει ενα από τα δύο ενδεχόμενα τοτε οι τιμές στις μεταβλητές in_bracket και in_parentheses αλλάζουν με τις κατάλληλες τιμές. Επίσης βάζει τον αριθμό γραμμής στην μεταβλητή line ελέγχει για τυχών χαρακτήρα κενού ή tab τα οποία και αγνοεί, αυτό γίνεται μέσα στον βρόκχο while.

Έπειτα συναντάμε το πρώτο switch του c το οποίο αναγνωρίζει την θέση των αγκύλων και των παρενθέσεων ενω αλλάζει την τιμη στην αναλογη μεταβλητη για το αν η αγκύλη ή η

παρένθεση που έχει εντοπιστεί ανοίγει που σε αυτή τη περίπτωση οι μεταβλητες παίρνουν την τιμη 1 (ενα), ενω παίρνουν την τιμή 0 (μηδέν) αν εντοπιστει κλειστει αγκύλη ή παρένθεση (αγκυλες: in_bracket και παρενθεσεις in_parentheses). Παρακάτω έχει χρησιμοποιηθεί ακόμα ένα switch το οποίο αναγνωρίζει και επιστρέφει στην οθόνη τα εξής το " , " επιστρεφει την λέξη COMMA, την " - " την αναγνωρίζει ως αριθμητική πράξη και επιστρέφει MINUS, το " + " ως αριθμητική πράξη και επιστρέφει PLUS, τον ειδικό χαρακτήρα αλλαγής γραμμής " \n " που επιστρέφει NEWLINE, τον ειδικό χαρακτήρα " EOF" κανει exit απο το πρόγραμμα, το σύμβολο " / " που αναγνωρίζεται ως αριθμητικη πράξη και επιστρέφει DIV, το συμβολο " * " που αναγνωρίζεται ως πολλαπλασιασμός και επιστρέφει MULTI, το "= " που το αναγνωριζει και επιστρέφει EQ, το " ; " που επιστρφει SEMI, τις " (" και ") " για τα οποία επιστρέφει PAR_START και PAR_END καθώς και τα " { " και " } " για τα οποια επιστρέφει BRA_START και BRA_END, τέλος το " ^ " ως αριθμητική πράξη και επιστρέφει POW. Για οτιδήποτε άλλο μπαίνει στο default της switch και καλει την yyerror η οποία εμφανίζει " invalid character". Έπειτα οριζετα η συνάρτηση yyerror και κλεινει το κομματι περιγραφής.

Τμήμα ορισμού Ε'(κυρίως πρόγραμμα)

Στο κυρίως πρόγραμμα δεν έχουμε πολλές εντολές παραμόνο την συνάρτηση yyparse απο το bison parser η οποία ξεκινάει την συντακτική ανάλυση.

Παραδείγματα λειτουργίας " simple-bison-code.y"

//παράδειγμα 1

open bracket{}

Scanner returned: VARIABLE (open)

Scanner returned: VARIABLE (bracket)

Στο παραδειγμα open bracket{} παρατηρουμε οτι το προγραμμα σωστα ανγνωριζει τις λεξεις ως μεταβλητές ενω τις αγκυλες τις αγνοεί.

//παράδειγμα 2

4+3=8

Scanner returned: INTCONST (4)

Scanner returned: PLUS (+)

Scanner returned: INTCONST (3)

Scanner returned: EQUALS (EQ)

Scanner returned: INTCONST (8)

Στο παραδειγμα με την αριθμητική πράξη 4+3=8 παρατειρούμε οτι το πρόγραμμα ανταποκρίνεται σωστα και διαβαζει τους αριθμούς ως intconst ενω τα συμβολτα αντιστοιχα με plus και equals.

//παράδειγμα 3

4=3/1 if \n

Scanner returned: INTCONST (4)

Scanner returned: EQUALS (EQ)

Scanner returned: INTCONST (3)

Scanner returned: DIVISION (DIV)

Scanner returned: INTCONST (1)

Scanner returned: VARIABLE (if)

Error: invalid character

Error: syntax error

Στο παράδειγμα 4=3/1 if \n το προγραμμα αντιστοιχεί σωστά στις κατηγορίες τους αριθμούς, παιρνει το

if σαν μεταβλητή και μόλις συναντάει το \n το αναγνωρίζει ως invalid character και βγαίνει απο το πρόγραμμα.

//παράδειγμα 4

6-3;

Scanner returned: INTCONST (6)

Scanner returned: MINUS (-)

Scanner returned: INTCONST (3)

Scanner returned: SEMICOLON (SEMI)

Error: syntax error

Στο παράδειγμα με είσοδο 6-3; ανταποκρίνεται σωστά σε ολα τα στοιχεία ακόμα και στο ερωτηματικό και τελειώνει με syntax error λογω του ερωτηματικού.

//παράδειγμα 5

int main (void); if(3>4)

Scanner returned: VARIABLE (int)

Scanner returned: VARIABLE (main)

Scanner returned: VARIABLE (void)

Scanner returned: SEMICOLON (SEMI)

Error: syntax error

Στο παράδειγμα int main (void); if(3>4) το προγραμμα με την αναγνώριση του ερωτηματικού τελειώνει το προγραμμα και δεν αναγνωριζει την συνεχεια του.

//παράδειγμα 6

int main void

Scanner returned: VARIABLE (int)

Scanner returned: VARIABLE (main)

Scanner returned: VARIABLE (void)

Σε αυτη τη περίπτωση το προγραμμα αναγνωρίζει και τις τρεις λεξεις σωστά σαν μεταβλητές.

//παράδειγμα 7

54@#^^5

Scanner returned: INTCONST (54)

Error: invalid character

Error: syntax error

Στο παράδειγμα 54@#^^5 αναγνωρίζει το 54 σαν αριθμο σωστα αλλα τελειώνει το προγραμμα λόγω αναγνώρισης ανεπιθύμητου χαρακτήρα που ειναι το " @ ".

//παράδειγμα 8

;

Scanner returned: SEMICOLON (SEMI)

Error: syntax error

Δίνοντας σαν εισοδο το ερωτηματικο και μονο βλέπουμε ότι το πρόγραμμα το αναγνωρίζει και εκτυπώνει semicolon και κλεινει το πρόγραμμα.

//παράδειγμα 9

cat ./simple-bison-code

Scanner returned: VARIABLE (cat)

Error: invalid character

Error: syntax error

Στο παραδειγμα με είσοδο cat ./simple-bison-code βλεπουμε οτι το πρόγραμμα αναγνωριζει την λεξη σαν μεταβλητη ενω βρήσει ανεπιθυμητο χαρακτηρα την " . " και εμφανιζει το μήνυμα, σταματωντας την λειτουργια.

//παράδειγμα 10

4-6=50/2

Scanner returned: INTCONST (4)

Scanner returned: MINUS (-)

Scanner returned: INTCONST (6)

Scanner returned: EQUALS (EQ)

Scanner returned: INTCONST (50)

Scanner returned: DIVISION (DIV)

Scanner returned: INTCONST (2)

Στο παραδειγμα με είσοδο 4-6=50/2 σαν αριθμητικη πραξη παρατηρουμε οτι το προγραμμα ανταποκρίνεται επιτύχως βαζοντας σε σωστες κατηγοριες ολα τα στοιχεία.

//παράδειγμα 11

4%4

Scanner returned: INTCONST (4)

Error: invalid character

Error: syntax error

Με εισοδο στο παράδειγμα το 4%4 το προγραμμα σταματαει αφου εντοπιζει ανεπιθυμητο χαρακτηρα και δεν συνεχιζει.

//παράδειγμα 12

4*5^5;

Scanner returned: INTCONST (4)

Scanner returned: MULTIPLY (MULTI)

Scanner returned: INTCONST (5)

Scanner returned: POWER (POW)

Scanner returned: INTCONST (5)

Στο παραδειγμα με είσοδο το 4*5^5 κατηγοριοποιεί τους αριθμους και ολα τα συμβολα σε σωστες κατηγοριες.

//παράδειγμα 13

{4*4}

Scanner returned: INTCONST (4)

Scanner returned: MULTIPLY (MULTI)

Scanner returned: INTCONST (4)

Στο παραδειγμα {4*4} το προγραμμα αγνοεί τις αγκύλες και κατηγοριοποιεί σωστα τους αριθμους και τις πράξεις.

//παράδειγμα 14

ooooooo*66=9

Error: integer starting with zero

Το προγραμμα ξεκιναει εντοπιζει οτι ο αριθμος ειναι 0 και τελειωνει το προγραμμα σε μήνυμα.

//παράδειγμα 15

01*45

Error: integer starting with zero

Το ίδιο συμβαίνει και εδω επειδη η πρωτη μεταβλητη ειναι 0 το προγραμμα σταματαει με μυνημα λαθους

//παράδειγμα 16

_23+2=2

Scanner returned: VARIABLE (_23)

Scanner returned: PLUS (+)

Scanner returned: INTCONST (2)

Scanner returned: EQUALS (EQ)

Scanner returned: INTCONST (2)

Στο παραπανω παραδειγμα το προγραμμα ανγνωρίζει το _23 σαν μεταβλητη και το κατηγοριοποιεί οπως και τα αλλα στοιχεια σε σωστες κατηγοριες.

//παράδειγμα 17

/n

Scanner returned: DIVISION (DIV)

Scanner returned: VARIABLE (n)

Στο παραδειγμα /n το προγραμμα το αναγνωριζει ως διαιρεση μεταβλητης το οποίο και βαζει σε κατηγορια division και variable.

//παράδειγμα 18

=+-_*

Scanner returned: EQUALS (EQ)

Scanner returned: PLUS (+)

Error: syntax error

Στο συγκεκριμένο παράδειγμα το προγραμμα σταματαει σε λαθος επειδη δεν υπαρχουν μεταβλητες δεξια και αριστερα απο το συμβολο της αφαιρεσης. Οπως και στο αμεσως επομενο παραδειγμα.

//παράδειγμα 19

* - + = / _

Scanner returned: MULTIPLY (MULTI)

Error: syntax error

//παράδειγμα 20

0-9+

Scanner returned: INTCONST (0)

Scanner returned: MINUS (-)

Scanner returned: INTCONST (9)

Scanner returned: PLUS (+)

Στην αριθμητικη πραξη 0-9+ το προγραμμα αναγνωριζει σωστα ολες τις κατηγοριες.

//παράδειγμα 21

8_7/2 left 5_6+1

Scanner returned: INTCONST (8)

Scanner returned: VARIABLE (_7)

Scanner returned: DIVISION (DIV)

Scanner returned: INTCONST (2)

Scanner returned: VARIABLE (left)

Scanner returned: INTCONST (5)

Scanner returned: VARIABLE (_6)

Scanner returned: PLUS (+)

Scanner returned: INTCONST (1)

Στο παραδειγμα 8_7/2 left 5_6+1 αναγνωριζει ολες τις κατηγοριες και τα συμβολα.

//παράδειγμα 22

x=4+5

Scanner returned: VARIABLE (x)

Scanner returned: EQUALS (EQ)

Scanner returned: INTCONST (4)

Scanner returned: PLUS (+)

Scanner returned: INTCONST (5)

Παραδείγματα λειτουργίας " bison-SA.y "

// παραδειγμα 1

cat input.txt | ./bison-SA

Line 1: Scanner returned token: KEYWORD_FUNC (func)

Line 1: Scanner returned token: ID (main)

Line 1: Scanner returned token: PAR_START (( )

Line 1: Scanner returned token: ID (argc)

Line 1: Scanner returned token: COMMA (,)

Line 1: Scanner returned token: ID (argv)

Line 1: Scanner returned token: PAR_END ( ))

Valid arguments

Valid function declaration!

Line 1: Scanner returned token: BRA_START ({)

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (metavliti)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (1)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (variable)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (0)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: ID (variable)

Line 1: Scanner returned token: PLUS + (+)

Line 1: Scanner returned token: INTCONST (1)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res2)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: ID (variable)

Line 1: Scanner returned token: MULTI (*)

Line 1: Scanner returned token: INTCONST (8)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res3)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: ID (variable)

Line 1: Scanner returned token: PLUS + (+)

Line 1: Scanner returned token: ID (metavliti)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res4)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: ID (variable)

Line 1: Scanner returned token: MINUS (-)

Line 1: Scanner returned token: INTCONST (1)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res5)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: PLUS + (+)

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res6)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (100)

Line 1: Scanner returned token: DIVISION (DIV)

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res7)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (700)

Line 1: Scanner returned token: MINUS (-)

Line 1: Scanner returned token: INTCONST (800)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res8)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (1)

Line 1: Scanner returned token: DIVISION (DIV)

Line 1: Scanner returned token: INTCONST (1)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res9)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (1)

Line 1: Scanner returned token: POW (^)

Line 1: Scanner returned token: INTCONST (100)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (res9)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (10)

Line 1: Scanner returned token: POW (^)

Line 1: Scanner returned token: ID (variable)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: BRA_END (})

Valid function body!

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: EOF (EOF)

Με την είσοδο του αρχείου το προγραμμα ανταποκρινεται σωστα και διαβαζει και κατηγοριοποιεί καθε λεξη και συμβολο.

//παράδειγμα 2

double positive=stay=5;

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (double)

Line 1: Scanner returned token: ID (positive)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: ID (stay)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Στο παραδειγμα με εισοδο double positive=stay=5; το προγραμμα αναγνωριζει ολες τις κατηγοριες, μας εμφανιζειι το μυνημα οτι η δηλωση ειναι εγκυρη και σταματαει το προγραμα με το χαρακτηρα \n.

//παράδειγμα 3

double positive= stay=4

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (double)

Line 1: Scanner returned token: ID (positive)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: ID (stay)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (4)

Line 2: Scanner returned token: NEWLINE (\n)

Error: syntax error

Στο προγραμμα με εισοδο double positive= stay=4 αναγνωριζονται ολα και στο τελος εμφανιζει μθνημα οτι υπαρχει προβλημα, αυτο γινεται διοτι στο τελος η εντολη δεν κλεινει με το ερωτηματικο.

//παράδειγμα 4

char x=5;

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (char)

Line 1: Scanner returned token: ID (x)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

Με την εισοδο char x=5; το προγραμμα αναγνωριζει ολα τα συμβολα και τις εντολες τα κατηγοριοποιει μας εμφανιζει μυνημα εγκυρης δηλωσης και σταματαει με τον χαρακτηρα αλλαγης γραμμης.

//παράδειγμα 5

double metabliti = t=5

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (double)

Line 1: Scanner returned token: ID (metabliti)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: ID (t)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (5)

Line 2: Scanner returned token: NEWLINE (\n)

Error: syntax error

Το παραδειγμα με αυτην την εισοδο ενω κατηγοριοποιει σωστα σταματαει μυνημα λαθους διοτι δεν υπαρχει ερωτηματικο να δηλωσει το τελος.

//παράδειγμα 6

double x=0,4=y;

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (double)

Line 1: Scanner returned token: ID (x)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (0)

Line 1: Scanner returned token: COMMA (,)

Error: syntax error

Το αποτελεσμα εμφανιζει μυνημα λαθους διοτι το κομμα δεν αναγνωριζεται.

//παράδειγμα 7

int met= 35+3;

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (met)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (35)

Line 1: Scanner returned token: PLUS + (+)

Line 1: Scanner returned token: INTCONST (3)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

//παράδειγμα 8

int met =4/5;

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (met)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (4)

Line 1: Scanner returned token: DIVISION (DIV)

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Line 2: Scanner returned token: NEWLINE (\n)

//παράδειγμα 9

;

Line 1: Scanner returned token: SEMI (;)

Error: syntax error

//παράδειγμα 10

5^2;

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: POW (^)

Line 1: Scanner returned token: INTCONST (2)

Line 1: Scanner returned token: SEMI (;)

Valid expression!

Line 2: Scanner returned token: NEWLINE (\n)

//παράδειγμα 11

int kapa = 151 + 2;

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (kapa)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (151)

Line 1: Scanner returned token: PLUS + (+)

Line 1: Scanner returned token: INTCONST (2)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Στο συγκεκριμένο παράδειγμα έχουμε ως keyword το int , ως ID το kapa ,το = το αναγνωρίζει ως EQ ορθά, πρόσθετα αναγνωρίζει και τις δύο ακέραιες τιμές 151 και 2 , επίσης παρατηρεί το + αναμεσά τους ως PLUS και το ; ως SEMI.

//παράδειγμα 12

int watkins = 75 - 32 ;

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (watkins)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (75)

Line 1: Scanner returned token: MINUS (-)

Line 1: Scanner returned token: INTCONST (32)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, έχουμε ως keyword το int , ως ID το watkins ,το = το αναγνωρίζει ως EQ ορθά, πρόσθετα αναγνωρίζει και τις δύο ακέραιες τιμές 75 και 32 , επίσης παρατηρεί το - αναμεσά τους ως MINUS και το ; ως SEMI.

//παράδειγμα 13

Locane = 32 * 4;

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: ID (locane)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (32)

Line 1: Scanner returned token: MULTI (*)

Line 1: Scanner returned token: INTCONST (4)

Line 1: Scanner returned token: SEMI (;)

Valid assignment!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, έχουμε ως ID το locane ,το = το αναγνωρίζει ως EQ ορθά, επιπρόσθετα αναγνωρίζει και τις δύο ακέραιες τιμές 32 και 4 ως INTCONST , επίσης παρατηρεί το * αναμεσά τους ως MULTI και το ; ως SEMI.

//παράδειγμα 14

int yp = 15 ^ 6;

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (yp)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (15)

Line 1: Scanner returned token: POW (^)

Line 1: Scanner returned token: INTCONST (6)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, έχουμε ως keyword το int , ως ID το yp ,το = το αναγνωρίζει ως EQ_ορθά, πρόσθετα αναγνωρίζει και τις δύο ακέραιες τιμές 15 και 6 , επίσης παρατηρεί το ^ αναμεσά τους ως POW μιας και μιλάμε για ύψωση σε δύναμη και το ; ως SEMI.

//παράδειγμα 15

Ital = 15 – 5;

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: ID (ital)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (15)

Line 1: Scanner returned token: MINUS (-)

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: SEMI (;)

Valid assignment!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, έχουμε ως ID το ital ,το = το αναγνωρίζει ως EQ_ορθά, επιπρόσθετα αναγνωρίζει και τις δύο ακέραιες τιμές 15 και 5 ως INTCONST και το ; ως SEMI.

//παράδειγμα 16

int jesus = 23;

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (jesus)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (23)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, έχουμε ως keyword το int , ως ID το jesus ,το = το αναγνωρίζει ως EQ και ορίζει μέσα στη μεταβλητή τον ακέραιο 23 με το semi ; στο τέλος

//παράδειγμα 17

int alah = 8;

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (alah)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (8)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, έχουμε ως keyword το int , ως ID το alah ,το = το αναγνωρίζει ως EQ , ορίζεται μέσα στη μεταβλητή τον ακέραιο 8 με το semi ; στο τέλος

//παράδειγμα 18

bouda = 32/6;

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: ID (bouda)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (32)

Line 1: Scanner returned token: DIVISION (DIV)

Line 1: Scanner returned token: INTCONST (6)

Line 1: Scanner returned token: SEMI (;)

Valid assignment!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, ως ID το bouda ,το = το αναγνωρίζει ως EQ , ορίζοντας μέσα στη μεταβλητή τον ακέραιο 32 και τον ακέραιο 6 καθώς και το / ως DIVISION με το semi ; στο τέλος.

//παράδειγμα 19

god = jesus + alah;

Line 2: Scanner returned token: NEWLINE (\n)

Line 1: Scanner returned token: ID (god)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: ID (jesus)

Line 1: Scanner returned token: PLUS + (+)

Line 1: Scanner returned token: ID (alah)

Line 1: Scanner returned token: SEMI (;)

Valid assignment!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, ως ID το god ,το = το αναγνωρίζει ως EQ , το jesus ως ID καθώς και το alah ως ID και ανάμεσα τους το + ως PLUS καθώς και ; ως SEMI.

//παράδειγμα 20

int k = 5;

Line 1: Scanner returned token: KEYWORD_VAR_TYPE (int)

Line 1: Scanner returned token: ID (k)

Line 1: Scanner returned token: EQ (=)

Line 1: Scanner returned token: INTCONST (5)

Line 1: Scanner returned token: SEMI (;)

Valid declaration!

Στο συγκεκριμένο παράδειγμα παρατηρεί το κενό ως NEWLINE, το int ως keyword καθώς αναγνωρίζεται και το κ ως ID , το = αναγνωρίζεται σωστά ως EQ, η τιμή 5 ως INTCONST και το ; ως SEMI.

Ευχαριστίες

"Ομαδα 15" εργαστηριακή άσκηση Β1 27/4/2021

ΣΑΣ ΕΥΧΑΡΙΣΤΟΥΜΕ ΠΟΛΥ ΓΙΑ ΤΟΝ ΧΡΟΝΟ ΣΑΣ