**Αρχές Γλωσσών Προγραμματισμού**

**Εργαστηριακή Άσκηση**

**Flex-Bison**

**Εγκατάσταση Flex και Bison σε Ubuntu VM**

Το Flex (Fast Lexical Analyzer Generator) είναι ένα εργαλείο για τη δημιουργία λεξικών αναλυτών, ενώ το Bison (GNU Bison) είναι μια γεννήτρια αναλυτών που παράγει συντακτικούς αναλυτές (parsers) με βάση μια τυπική προδιαγραφή γραμματικής.

To Flex εγκατίσταται μέσω της εντολής

**sudo apt-get install flex**



A screenshot of a computer program

Description automatically generated with medium confidence

Αντίστοιχα για το Bison:

**sudo apt install bison**

**A screenshot of a computer

Description automatically generated with medium confidence**

Τώρα που εγκαταστάθηκαν τα πακέτα Flex και Bison είναι δυνατόν να τρέξουν με αρχείο εισόδου που είναι γραμμένο στην ψευδογλώσσα που περιγράφεται από τους κανόνες στην εκφώνηση της εργαστηριακή άσκησης.

**Ερώτημα 1**

**Γραμματική ΒΝF**

<program> ::= {<class>}

<class> ::= "public class" <class\_name> "{" [<variable\_declaration>] [<method\_declaration>] "}"

<class\_name> ::= <uppercase\_letter> {<letter\_or\_digit>}

<variable\_declaration> ::= [<modifier>] <data\_type> <variable\_name> ";"

<modifier> ::= "public" | "private"

<data\_type> ::= "int" | "char" | "double" | "boolean" | "String"

<variable\_name> ::= <letter> {<letter\_or\_digit>}

<method\_declaration> ::= {<modifier>} [<return\_type>] <method\_name> "(" [<parameter\_list>] ")" "{" [<variable\_declaration>] <statements> "}"

<return\_type> ::= <data\_type> | "void"

<method\_name> ::= <lowercase\_letter> {<letter\_or\_digit>}

<parameter\_list> ::= [<parameter> {"," <parameter>}]

<parameter> ::= <data\_type> <parameter\_name>

<parameter\_name> ::= <lowercase\_letter> {<letter\_or\_digit>}

<statements> ::= {<statement>}

<statement> ::= <assignment\_statement> | <loop\_statement> | <control\_statement> | <print\_statement> | <return\_statement> | <break\_statement>

<assignment\_statement> ::= <variable\_name> "=" <expression> ";"

<loop\_statement> ::= "do" "{" <statements> "}" "while" "(" <expression> ");" | "for" "(" <expression> ";" <expression> ";" <expression> ")" "{" <statements> "}"

<control\_statement> ::= "if" "(" <expression> ")" "{" <statements> "}" {"else if" "(" <expression> ")" "{" <statements> "}"} ["else" "{" <statements> "}"] | "switch" "(" <expression> ")" "{" { "case" <expression> ":" <statements> } "default" ":" <statements> "}"

<print\_statement> ::= "out.print(" <string\_literal> ["," <variable\_name>] ");"

<return\_statement> ::= "return" [<expression>] ";"

<break\_statement> ::= "break" ";"

<expression> ::= <term> {("+" | "-" | "\*" | "/") <term>}

<term> ::= <variable\_name> | <literal> | <method\_call> | <object\_creation>

<literal> ::= <numeric\_literal> | <char\_literal> | <boolean\_literal>

<numeric\_literal> ::= <digit> {"." <digit>} "d"

<char\_literal> ::= "'" <character> "'"

<boolean\_literal> ::= "true" | "false"

<method\_call> ::= <object\_name> "." <method\_name> "(" [<argument\_list>] ")"

<object\_name> ::= <variable\_name>

<argument\_list> ::= [<expression> {"," <expression>}]

<object\_creation> ::= <class\_name> <object\_name> "=" "new" <class\_name> "(" ")" ";"

<string\_literal> ::= "\"" {<any\_character>} "\""

<any\_character> ::= <letter> | <digit> | <special\_character>

<letter\_or\_digit> ::= <letter> | <digit>

<letter> ::= <uppercase\_letter> | <lowercase\_letter>

<uppercase\_letter> ::= "A" | "B" | ... | "Z"

<lowercase\_letter> ::= "a" | "b" | ... | "z"

<digit> ::= "0" | "1" | ... | "9"

<special\_character> ::= "\_" | "+" | "-" | "\*" | "/" | "." | "," | ";" | ":" | "(" | ")" | "{" | "}" | "[" | "]" | "<" | ">" | "=" | "!" | "&" | "|" | "'" | "\"" | " " | "\n"

Σε αυτή την ψευδογλώσσα, ένα πρόγραμμα αποτελείται από μία ή περισσότερες κλάσεις, καθεμία από τις οποίες ορίζεται από τη λέξη-κλειδί «public class» ακολουθούμενη από το όνομα της κλάσης και περικλείεται μέσα σε αγκύλες. Τα ονόματα κλάσεων πρέπει να ξεκινούν με κεφαλαίο γράμμα και μπορούν να ακολουθούνται από οποιονδήποτε συνδυασμό γραμμάτων ή ψηφίων. Οι κλάσεις μπορούν να περιλαμβάνουν προαιρετικές δηλώσεις μεταβλητών και δηλώσεις μεθόδων. Οι δηλώσεις μεταβλητών αποτελούνται από έναν προαιρετικό τροποποιητή (είτε «public» είτε «private»), ακολουθούμενο από έναν τύπο δεδομένων και ένα όνομα μεταβλητής, το οποίο ολοκληρώνεται με άνω και κάτω τελεία. Οι δηλώσεις μεθόδων, ακολουθώντας σύνταξη τύπου Java, μπορούν να περιλαμβάνουν προαιρετικούς τροποποιητές, έναν τύπο επιστροφής (είτε έναν τύπο δεδομένων είτε «void»), ένα όνομα μεθόδου, έναν κατάλογο παραμέτρων και το σώμα της μεθόδου, το οποίο περιλαμβάνει δηλώσεις μεταβλητών και δηλώσεις.

Η γραμματική ορίζει επίσης διάφορες δηλώσεις, όπως δηλώσεις ανάθεσης, βρόχου, ελέγχου, εκτύπωσης, επιστροφής και διακοπής, μαζί με εκφράσεις που αποτελούνται από όρους που συνδέονται με αριθμητικούς τελεστές. Οι όροι μπορεί να είναι μεταβλητές, literals, κλήσεις μεθόδων ή δημιουργίες αντικειμένων. Οι κυριολεκτικές τιμές μπορεί να είναι αριθμητικές, χαρακτήρες ή boolean, ενώ οι κλήσεις μεθόδων και οι δημιουργίες αντικειμένων ακολουθούν συγκεκριμένα μοτίβα με ονόματα αντικειμένων και μεθόδων. Η BNF περιλαμβάνει ολόκληρη τη δομή αυτής της ψευδογλώσσας, παρέχοντας έναν ολοκληρωμένο οδηγό για την ανάλυση και την ερμηνεία του συντακτικού της.

**Υλοποίηση Flex + Bison**

%{

#include <stdio.h>

#include "parser.tab.h"

%}

%option noyywrap

/\* Integrated Μεταβλητή που μετράει τις καινούργιες γραμμές στο αρχείο εισόδου και αυξάνει έναν μετρητή \*/

%option yylineno

/\* Στην περίπτωση αυτή, το %x COMMENT δηλώνει μια συνθήκη έναρξης με το όνομα COMMENT, η οποία ενεργοποιείται όταν ο λεξικογράφος συναντήσει την αρχή ενός σχολίου στην είσοδο. \*/

%x COMMENT

%%

"public"                { printf("Variable: %s\n", yytext);return PUBLIC; }

"private"               { return PRIVATE; }

"void"                  { return VOID; }

"int"                   { return INT; }

"char"                  { return CHAR; }

"double"                { return DOUBLE; }

"boolean"               { return BOOLEAN; }

"String"                { return STRING; }

"class"                 { return CLASS; }

"new"                   { return NEW; }

"return"                { return RETURN; }

"if"                    { return IF; }

"else"                  { return ELSE; }

"while"                 { return WHILE; }

"do"                    { return DO; }

"for"                   { return FOR; }

"switch"                { return SWITCH; }

"case"                  { return CASE; }

"default"               { return DEFAULT; }

"break"                 { return BREAK; }

"true"                  { return TRUE; }

"false"                 { return FALSE; }

"out.print"             { return OUTPRINT; }

"out.println"           { return OUTPRINTLN; }

"{"                     { printf("Variable: %s\n", yytext);return RIGHT\_BRC; }

"}"                     { printf("Variable: %s\n", yytext);return LEFT\_BRC; }

"("                     { return RIGHT\_PAR; }

")"                     { return LEFT\_PAR; }

";"                     { return EROTIMATIKO; }

":"                     { return COLON; }

","                     { return COMMA; }

"="                     { return EQUALS; }

\"                   { return QT; }

"=="                    { return EQ\_OP; }

"!="                    { return NE\_OP; }

">"                     { return GREATER\_THAN; }

"<"                     { return LESS\_THAN; }

">="                    { return GE\_OP; }

"<="                    { return LE\_OP; }

"+"                     { return PLUS; }

"-"                     { return MINUS; }

"\*"                     { return STAR; }

"/"                     { return SLASH; }

"&&"                    { return AND\_OP; }

"||"                    { return OR\_OP; }

"//".\*                  { /\* Ignore single-line comments \*/ }

"/\*"([^\*]|(\\*+[^\*/]))\*"\*/" { /\* Ignore multi-line comments \*/ }

[ \t\n\r]+              { /\* Ignore whitespace \*/ }

[A-Za-z][\_A-Za-z0-9]\*    { printf("Variable: %s\n", yytext); return T\_VAR; }

   /\* Κανονική Έκφραση για ακέραιο \*/

[0-9]+                   { yylval = atoi(yytext); return T\_INTEGER; }

   /\* Κανονική Έκφραση για δεκαδικό \*/

[+-]?[0-9]+\.[0-9]+     { return T\_DOUBLE; }

%%

Το πρώτο τμήμα κώδικα του Flex περιλαμβάνει κώδικα C. Στο σημείο αυτό, γίνεται include η βιβλιοθήκη <stdio.h> της C και επίσης προκειμένου να συμπεριληφθούν τα headers που δημιουργεί το bison στον λεκτικό αναλυτή (parser.tab.h) γίνεται import to parser.tab.h. Έπειτα, προστέθηκε το %option noyywrap που είναι ένα option του Flex API στο οποίο ενημερώνει να μην καλέσει τη yywrap όταν φτάσει στο τέλος του input file.

Έπειτα, στο επόμενο block κώδικα δηλώνονται τα σύμβολα (tokens) που διαβάζει το flex από το αρχείο εισόδου.

Τα δεσμευμένα tokens πρώτα όπως τα παρακάτω:

"public" { printf("Variable: %s\n", yytext);return PUBLIC; }

"private" { return PRIVATE; }

"void" { return VOID; }

"int" { return INT; }

"char" { return CHAR; }

Με το που αναγνωριστούν από τον flex επιστρέφονται οι παραπάνω μεταβλητές. Επιπλέον, για περιπτώσεις debugging έχουν προστεθέι εντολές print προκειμένου να φαίνεται στο τερματικό η αναγνώριση από flex.

Τέλος, δηλώνονται οι κανονικές εκφράσεις για να αναγνωρίζονται οι τύποι δεδομένων όπως Ακέραιοι, Strings, Double   
  
[A-Za-z][\_A-Za-z0-9]\* { printf("Variable: %s\n", yytext); return T\_VAR; }

/\* Κανονική Έκφραση για ακέραιο \*/

[0-9]+ { yylval = atoi(yytext); return T\_INTEGER; }

/\* Κανονική Έκφραση για δεκαδικό \*/

[+-]?[0-9]+\.[0-9]+ { return T\_DOUBLE; }

**Ο κώδικας για το Bison:**

%{

    /\* Το αρχείο Bison  -> parser.y

    Στο πρώτο τμήμα του ορίζεται ο κώδικας C\*/

    #include <stdio.h>

    #include <stdlib.h>

    #include <unistd.h>

extern FILE \*yyin; // Εξωτερική δήλωση για το αρχείο εισόδου

extern int yylex(); // Εξωτερική δήλωση για τη συνάρτηση λεκτικής ανάλυσης

extern char \*yytext; // Εξωτερική δήλωση για τη μεταβλητή yytext

extern int yylineno; // Εξωτερική δήλωση για τη μεταβλητή yylineno

int error\_count = 0; // Μεταβλητή για την καταγραφή του αριθμού των σφαλμάτων

int flag\_err\_type = 0; // Μεταβλητή για τον έλεγχο του είδους του σφάλματος

int scope = 0; // Μεταβλητή για το επίπεδο εμβέλειας των μεταβλητών

void yyerror(const char \*message); // Πρότυπο για τη συνάρτηση διαχείρισης σφαλμάτων

    %}

%token PUBLIC PRIVATE VOID INT CHAR DOUBLE BOOLEAN STRING CLASS NEW RETURN IF ELSE WHILE DO FOR SWITCH CASE DEFAULT BREAK TRUE FALSE OUTPRINT OUTPRINTLN

%token EQ\_OP COLON NE\_OP GE\_OP LE\_OP AND\_OP OR\_OP EROTIMATIKO COMMA EQUALS GREATER\_THAN LESS\_THAN PLUS MINUS STAR SLASH

%token T\_VAR T\_INTEGER T\_DOUBLE T\_STR T\_INTEGER\_STRING LEFT\_PAR RIGHT\_PAR LEFT\_BRC RIGHT\_BRC QT

%start program

// Grammar rules

%%

program:

    class\_list

    ;

class\_list:

    class\_list class\_decl

    | class\_decl

    ;

class\_decl:

    PUBLIC CLASS identifier RIGHT\_BRC var\_decls method\_decls LEFT\_BRC

    ;

method\_decls:

    method\_decls method\_decl

    | method\_decl

    | %empty

    ;

method\_decl:

    modifier type identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    | identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    | modifier VOID identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    ;

var\_decls:

    var\_decls var\_decl

    | var\_decl

    | %empty

    ;

var\_decl:

    type identifier EROTIMATIKO

    | modifier type identifier EROTIMATIKO

    ;

modifier:

    PUBLIC

    | PRIVATE

    ;

type:

    INT

    | CHAR

    | DOUBLE

    | BOOLEAN

    | STRING

    ;

param\_list:

    param\_list COMMA param

    | param

    | %empty

    ;

param:

    type identifier

    ;

opt\_statements:

    statement\_list

    | %empty /\* empty \*/

    ;

statement\_list:

    statement\_list statement

    | statement

    ;

statement:

    var\_decl

    | assignment

    | if\_statement

    | while\_statement

    | do\_while\_statement

    | for\_statement

    | switch\_statement

    | out\_print

    | return\_statement

    | break\_statement

    ;

assignment:

    identifier EQUALS expression EROTIMATIKO

    ;

if\_statement:

    IF RIGHT\_PAR expression LEFT\_PAR RIGHT\_BRC opt\_statements LEFT\_BRC opt\_else

    ;

opt\_else:

    ELSE RIGHT\_BRC opt\_statements LEFT\_BRC

    | %empty /\* empty \*/

    ;

while\_statement:

    WHILE RIGHT\_PAR expression LEFT\_PAR RIGHT\_PAR opt\_statements LEFT\_PAR

    ;

do\_while\_statement:

    DO RIGHT\_BRC opt\_statements LEFT\_BRC WHILE RIGHT\_PAR expression LEFT\_PAR EROTIMATIKO

    ;

for\_statement:

    FOR RIGHT\_PAR type assignment expression EROTIMATIKO assignment LEFT\_PAR RIGHT\_BRC opt\_statements LEFT\_BRC

    ;

switch\_statement:

    SWITCH RIGHT\_PAR expression LEFT\_PAR RIGHT\_BRC case\_list opt\_default\_case LEFT\_BRC

    ;

case\_list:

    case\_list case\_clause

    | case\_clause

    ;

case\_clause:

    CASE expression COLON opt\_statements

    ;

opt\_default\_case:

    DEFAULT COLON opt\_statements

    | %empty /\* empty \*/

    ;

out\_print:

    OUTPRINT RIGHT\_PAR QT var\_list QT LEFT\_PAR EROTIMATIKO

    | OUTPRINTLN RIGHT\_PAR QT var\_list QT LEFT\_PAR EROTIMATIKO

    ;

var\_list:

    var\_list T\_VAR

    | T\_VAR

    ;

return\_statement:

    RETURN expression EROTIMATIKO

    | RETURN EROTIMATIKO

    ;

break\_statement:

    BREAK EROTIMATIKO

    ;

expression:

    expression PLUS expression

    | expression MINUS expression

    | expression STAR expression

    | expression SLASH expression

    | expression GREATER\_THAN expression

    | expression LESS\_THAN expression

    | expression GE\_OP expression

    | expression LE\_OP expression

    | expression EQ\_OP expression

    | expression NE\_OP expression

    | expression AND\_OP expression

    | expression OR\_OP expression

    | RIGHT\_PAR expression LEFT\_PAR

    | identifier

    | T\_INTEGER

    | T\_DOUBLE

    | T\_STR

    ;

identifier:

    T\_VAR

    ;

%%

int main(int argc, char \*argv[]){

    /\* If statement για να ελέγξει το αρχείο εισόδου \*/

    if(argc > 1){

        yyin = fopen(argv[1], "r");

        if (yyin == NULL){

            perror ("Σφάλμα ανοίγοντας το αρχείο"); return -1;

        }

    }

    /\* Καλείται η συνάρτηση yyparse που επιστρέφει 0 αν το input einai έγκυρο

         και 1 εάν υπάρχει συντακικό λάθος\*/

    yyparse();

    /\* Κλείσιμο αρχείου εισόδου\*/

    fclose(yyin);

    if(error\_count > 0){

        printf("Η συντακτική ανάλυση απέτυχε! Βρέθηκαν %d σφάλματα\n", error\_count);

    }else{

        printf("Η συντακτική ανάλυση ολοκληρώθηκε με επιτυχία! \nΔεν βρέθηκαν σφάλματα!\n");

    }

    return 0;

}

void yyerror(const char \*message)

{

    /\* Μετρητής για αριθμό σφαλμάτων \*/

    error\_count++;

    if(flag\_err\_type==0){

        printf("-> Σφάλμα στη γραμμή %d που προέκυψε από %s : %s\n", yylineno, yytext, message);

    }

    flag\_err\_type = 0; // Reset flag\_err\_type

}

Όπως και στο Flex, το πρώτο τμήμα του Bison περιλαμβάνει κώδικα C στον οποίο αρχικά δηλώνονται οι βιβλιοθήκες. Οι εξωτερικές δηλώσεις FILE \*yyin, int yylex(), char \*yytext, int yylineno χρησιμοποιούνται για την αναγνώριση και την ανάλυση του αρχείου εισόδου. Η μεταβλητή error\_count καταγράφει τον αριθμό των σφαλμάτων που έχουν ανιχνευθεί κατά τη διάρκεια της ανάλυσης, ενώ η μεταβλητή flag\_err\_type χρησιμοποιείται για τον έλεγχο του είδους του σφάλματος που εντοπίστηκε. Η μεταβλητή scope αντιπροσωπεύει το επίπεδο εμβέλειας των μεταβλητών στον κώδικα. Τέλος, η συνάρτηση yyerror χρησιμοποιείται για την εκτύπωση μηνυμάτων λάθους και τη διαχείριση των σφαλμάτων κατά τη διάρκεια της εκτέλεσης του προγράμματος.

Στο επόμενο τμήμα κώδικα του συντακτικού αναλυτή Bison δηλώνονται τα tokens που εντοπίζει στο Flex προκειμένου να χρησιμοποιηθούν σε κανόνες της γραμματικής. Όσον αφορά τη γραμματική του Bison, έχουν οριστεί όλοι οι κανόνες που αναφέρονται στην εκφώνηση καθώς και στην BNF γραμματική.

Η παρεχόμενη γραμματική Bison ορίζει τη δομή μιας γλώσσας προγραμματισμού, περιλαμβάνοντας κλάσεις, μεθόδους, μεταβλητές, εντολές ροής ελέγχου και εκφράσεις. Ο κανόνας `**program**` χρησιμεύει ως σημείο εκκίνησης, υποδεικνύοντας ότι ένα πρόγραμμα αποτελείται από μια λίστα κλάσεων. Κάθε κλάση στη `class\_list` μπορεί να δηλωθεί είτε μεμονωμένα είτε διαδοχικά, με κάθε δήλωση κλάσης (`**class\_decl**`) να περιλαμβάνει τροποποιητές ορατότητας, ονόματα κλάσεων, δηλώσεις μεταβλητών (`**var\_decls**`) και δηλώσεις μεθόδων (`**method\_decls**`). Οι μέθοδοι μέσα σε μια κλάση μπορούν να οριστούν με ή χωρίς τύπους επιστροφής (`**type**`) και μπορούν να έχουν παραμέτρους (`**param\_list**`) και δηλώσεις τοπικών μεταβλητών (`**var\_decls**`). Περιλαμβάνονται δηλώσεις ροής ελέγχου όπως `**if**`, `**while**`, `**do-while**`, `**fo**r` και `**switch**`, καθώς και δηλώσεις για εκχωρήσεις μεταβλητών, κλήσεις μεθόδων (`**out\_print**`), επιστροφές και διακοπές.

Η αρθρωτότητα και η αναδρομικότητα της γραμματικής επιτρέπουν ευέλικτες δομές και δηλώσεις προγραμμάτων. Χειρίζεται διάφορα σενάρια, όπως μεθόδους χωρίς τύπους επιστροφής (`**VOID**`), κενές λίστες παραμέτρων ή μεταβλητών και προαιρετικές διακλαδώσεις `else` σε δηλώσεις `if`. Η γραμματική ορίζει επίσης εκφράσεις με τελεστές (`**PLUS**`, `**MINUS**`, κ.λπ.), παρενθέσεις, αναγνωριστικά και κυριολεκτικά (`**T\_INTEGER**`, `**T\_DOUBLE**`, `**T\_STR**`).

**Δοκιμή του Compiler με Input File**

Για το αρχείο εισόδου ορίστηκε το αρχείο **sample.txt**

public class A {

int var1;

private double var2;

public int method () {

private double var3;

out.println("Hello world!");

private String var4;

var1=5;

var4 = var7 + 1;

for (int i = 0; i < 5; i = i+1;) {

out.println("Hello world!");

}

do {

out.println("Hello world!");

num= num+1;

} while (num != 5);

if (num == 10){

private String str;

}

//This is a comment

/\* This

is

a

multi line

comment

\*/

switch(a) {

case 1:

out.println("case\_1");

break;

case 2:

out.println("case2");

default:

out.println("defaultt");

}}}

Στο αρχείο αυτό υπάρχουν όλα τα χαρακτηριστικά της γραμματικής.   
Δηλώσεις κλάσεων, μεθόδων, μεταβλητών, for loops, if statements, σχόλια multiline και single line.

Στο τερματικό ubuntu τρέχουμε την εντολή για να γίνει compile to Flex αρχείο (flex.l).

**flex flex.l**

Βλέπουμε ότι δημιουργείται το αρχείο 

Έπειτα, χρειάζεται να γίνει compiled o bison **bison -d parser.y**

Βλέπουμε οτι δημιουργούντα τα αρχεία A picture containing text, font, screenshot, graphics

Description automatically generated

Τέλος, εκτελούμε με κώδικα C τα παραγόμενα αρχεία του flex και του bison προκειμένου να δημιουργηθεί ο compiler (parser).

**gcc -o parser lex.yy.c parser.tab.c -lfl**

Τέλος, γίνεται compile το αρχείο εισόδου example.xml και παρατηρούμε τα αποτελέσματα.

**./parser sample.txt**

**A black background with white text

Description automatically generated**

Φαίνεται οτι το compiler αναγνωρίζει με επιτυχία το αρχείο.

Τώρα γίνεται και η δοκιμή για μια ανεπιτυχή ανάλυση.

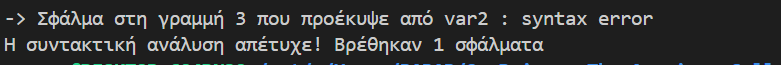
Στη γραμμη 3 του αρχείου εισόδου sample.txt

Αφαιρούμε το type double από τον ορισμό της μεταβλητής

private var2;

A computer screen shot of a black screen

Description automatically generated

Εκτελούμε τον parser και εμφανίζεται το Σφάλμα :   
  


Ο οποίος επιστρέφει το σφάλμα και αναφέρει οτι προέκυψε στη γραμμή 3 επειδή αφαιρέθηκε το type της μεταβλητής var2. Επίσης, αναφέρει ότι προήλθε από τη μεταβλτητή var2 και οτι πρόκειται για συντακτικό λάθος.

Έπειτα, μια ακόμη δοκιμή που κάναμε είναι να βάλουμε τα σχόλια :  
A screenshot of a computer

Description automatically generated



**Ερώτημα 2**

Το νέο αρχείο Bison με τις αλλαγές ώστε να να αναγνωρίζεται ανάθεση τιμής κατά τη δήλωση μιας μεταβλητής και η δήλωση πολλαπλών μεταβλητών ίδιου τύπου σε μια γραμμή.

%{

    /\* Το αρχείο Bison  -> parser.y

    Στο πρώτο τμήμα του ορίζεται ο κώδικας C\*/

    #include <stdio.h>

    #include <stdlib.h>

    #include <unistd.h>

extern FILE \*yyin; // Εξωτερική δήλωση για το αρχείο εισόδου

extern int yylex(); // Εξωτερική δήλωση για τη συνάρτηση λεκτικής ανάλυσης

extern char \*yytext; // Εξωτερική δήλωση για τη μεταβλητή yytext

extern int yylineno; // Εξωτερική δήλωση για τη μεταβλητή yylineno

int error\_count = 0; // Μεταβλητή για την καταγραφή του αριθμού των σφαλμάτων

int flag\_err\_type = 0; // Μεταβλητή για τον έλεγχο του είδους του σφάλματος

int scope = 0; // Μεταβλητή για το επίπεδο εμβέλειας των μεταβλητών

void yyerror(const char \*message); // Πρότυπο για τη συνάρτηση διαχείρισης σφαλμάτων

    %}

%token PUBLIC PRIVATE VOID INT CHAR DOUBLE BOOLEAN STRING CLASS NEW RETURN IF ELSE WHILE DO FOR SWITCH CASE DEFAULT BREAK TRUE FALSE OUTPRINT OUTPRINTLN

%token EQ\_OP COLON NE\_OP GE\_OP LE\_OP AND\_OP OR\_OP EROTIMATIKO COMMA EQUALS GREATER\_THAN LESS\_THAN PLUS MINUS STAR SLASH

%token T\_VAR T\_INTEGER T\_DOUBLE T\_STR T\_INTEGER\_STRING LEFT\_PAR RIGHT\_PAR LEFT\_BRC RIGHT\_BRC QT

%start program

// Grammar rules

%%

program:

    class\_list

    ;

class\_list:

    class\_list class\_decl

    | class\_decl

    ;

class\_decl:

    PUBLIC CLASS identifier RIGHT\_BRC var\_decls method\_decls LEFT\_BRC

    ;

method\_decls:

    method\_decls method\_decl

    | method\_decl

    | %empty

    ;

method\_decl:

    modifier type identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    | identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    | modifier VOID identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    ;

var\_decls:

    var\_decls var\_decl

    | var\_decl

    | %empty

    ;

var\_decl:

    type var\_decl\_list EROTIMATIKO

    | modifier type var\_decl\_list EROTIMATIKO

    ;

var\_decl\_list:

    var\_decl\_list COMMA var\_assign

    | var\_assign

    ;

var\_assign:

    identifier EQUALS expression

    | identifier

    ;

modifier:

    PUBLIC

    | PRIVATE

    ;

type:

    INT

    | CHAR

    | DOUBLE

    | BOOLEAN

    | STRING

    ;

param\_list:

    param\_list COMMA param

    | param

    | %empty

    ;

param:

    type identifier

    ;

opt\_statements:

    statement\_list

    | %empty /\* empty \*/

    ;

statement\_list:

    statement\_list statement

    | statement

    ;

statement:

    var\_decl

    | assignment

    | if\_statement

    | while\_statement

    | do\_while\_statement

    | for\_statement

    | switch\_statement

    | out\_print

    | return\_statement

    | break\_statement

    ;

assignment:

    identifier EQUALS expression EROTIMATIKO |

    identifier EQUALS expression

    ;

if\_statement:

    IF RIGHT\_PAR expression LEFT\_PAR RIGHT\_BRC opt\_statements LEFT\_BRC opt\_else

    ;

opt\_else:

    ELSE RIGHT\_BRC opt\_statements LEFT\_BRC

    | %empty /\* empty \*/

    ;

while\_statement:

    WHILE RIGHT\_PAR expression LEFT\_PAR RIGHT\_PAR opt\_statements LEFT\_PAR

    ;

do\_while\_statement:

    DO RIGHT\_BRC opt\_statements LEFT\_BRC WHILE RIGHT\_PAR expression LEFT\_PAR EROTIMATIKO

    ;

for\_statement:

    FOR RIGHT\_PAR type assignment expression EROTIMATIKO assignment LEFT\_PAR RIGHT\_BRC opt\_statements LEFT\_BRC

    ;

switch\_statement:

    SWITCH RIGHT\_PAR expression LEFT\_PAR RIGHT\_BRC case\_list opt\_default\_case LEFT\_BRC

    ;

case\_list:

    case\_list case\_clause

    | case\_clause

    ;

case\_clause:

    CASE expression COLON opt\_statements

    ;

opt\_default\_case:

    DEFAULT COLON opt\_statements

    | %empty /\* empty \*/

    ;

out\_print:

    OUTPRINT RIGHT\_PAR QT var\_list QT LEFT\_PAR EROTIMATIKO

    | OUTPRINTLN RIGHT\_PAR QT var\_list QT LEFT\_PAR EROTIMATIKO

    ;

var\_list:

    var\_list T\_VAR

    | T\_VAR

    ;

return\_statement:

    RETURN expression EROTIMATIKO

    | RETURN EROTIMATIKO

    ;

break\_statement:

    BREAK EROTIMATIKO

    ;

expression:

    expression PLUS expression

    | expression MINUS expression

    | expression STAR expression

    | expression SLASH expression

    | expression GREATER\_THAN expression

    | expression LESS\_THAN expression

    | expression GE\_OP expression

    | expression LE\_OP expression

    | expression EQ\_OP expression

    | expression NE\_OP expression

    | expression AND\_OP expression

    | expression OR\_OP expression

    | RIGHT\_PAR expression LEFT\_PAR

    | identifier

    | T\_INTEGER

    | T\_DOUBLE

    | T\_STR

    ;

identifier:

    T\_VAR

    ;

%%

int main(int argc, char \*argv[]){

    /\* If statement για να ελέγξει το αρχείο εισόδου \*/

    if(argc > 1){

        yyin = fopen(argv[1], "r");

        if (yyin == NULL){

            perror ("Σφάλμα ανοίγοντας το αρχείο"); return -1;

        }

    }

    /\* Καλείται η συνάρτηση yyparse που επιστρέφει 0 αν το input einai έγκυρο

         και 1 εάν υπάρχει συντακικό λάθος\*/

    yyparse();

    /\* Κλείσιμο αρχείου εισόδου\*/

    fclose(yyin);

    if(error\_count > 0){

        printf("Η συντακτική ανάλυση απέτυχε! Βρέθηκαν %d σφάλματα\n", error\_count);

    }else{

        printf("Η συντακτική ανάλυση ολοκληρώθηκε με επιτυχία! \nΔεν βρέθηκαν σφάλματα!\n");

    }

    return 0;

}

void yyerror(const char \*message)

{

    /\* Μετρητής για αριθμό σφαλμάτων \*/

    error\_count++;

    if(flag\_err\_type==0){

        printf("-> Σφάλμα στη γραμμή %d που προέκυψε από %s : %s\n", yylineno, yytext, message);

    }

    flag\_err\_type = 0; // Reset flag\_err\_type

}

**var\_decl** Ενημέρωση κανόνα:

Μπορεί να χειριστεί πολλαπλές δηλώσεις μεταβλητών.

Υποστηρίζει τον προαιρετικό τροποποιητή πριν από τον τύπο.

**var\_decl\_list** Κανόνας:

Αυτός ο κανόνας εισάγεται για τη διαχείριση μιας λίστας δηλώσεων μεταβλητών που χωρίζονται με κόμμα.

**var\_assign** Κανόνας:

Αυτός ο κανόνας διαχειρίζεται την ανάθεση τιμών σε μεταβλητές και τη δήλωση μεταβλητών χωρίς αρχική τιμή.Υποστηρίζει τόσο την έκφραση identifier EQUALS όσο και το identifier.

Η Νεα BNF γραμματική που προκύπτει είναι η εξής :   
  
<program> ::= {<class>}

<class> ::= "public class" <class\_name> "{" [<variable\_declarations>] [<method\_declarations>] "}"

<class\_name> ::= <uppercase\_letter> {<letter\_or\_digit>}

<variable\_declarations> ::= {<variable\_declaration>}

<variable\_declaration> ::= [<modifier>] <data\_type> <variable\_assignments> ";"

<variable\_assignments> ::= <variable\_assignment> {"," <variable\_assignment>}

<variable\_assignment> ::= <variable\_name> ["=" <expression>]

<modifier> ::= "public" | "private"

<data\_type> ::= "int" | "char" | "double" | "boolean" | "String"

<variable\_name> ::= <letter> {<letter\_or\_digit>}

<method\_declarations> ::= {<method\_declaration>}

<method\_declaration> ::= [<modifier>] [<return\_type>] <method\_name> "(" [<parameter\_list>] ")" "{" [<variable\_declarations>] <statements> "}"

<return\_type> ::= <data\_type> | "void"

<method\_name> ::= <lowercase\_letter> {<letter\_or\_digit>}

<parameter\_list> ::= [<parameter> {"," <parameter>}]

<parameter> ::= <data\_type> <parameter\_name>

<parameter\_name> ::= <lowercase\_letter> {<letter\_or\_digit>}

<statements> ::= {<statement>}

<statement> ::= <assignment\_statement> | <loop\_statement> | <control\_statement> | <print\_statement> | <return\_statement> | <break\_statement> | <variable\_declaration>

<assignment\_statement> ::= <variable\_name> "=" <expression> ";"

<loop\_statement> ::= "do" "{" <statements> "}" "while" "(" <expression> ");" | "for" "(" <expression> ";" <expression> ";" <expression> ")" "{" <statements> "}"

<control\_statement> ::= "if" "(" <expression> ")" "{" <statements> "}" {"else if" "(" <expression> ")" "{" <statements> "}"} ["else" "{" <statements> "}"] | "switch" "(" <expression> ")" "{" { "case" <expression> ":" <statements> } "default" ":" <statements> "}"

<print\_statement> ::= "out.print(" <string\_literal> ["," <variable\_name>] ");"

<return\_statement> ::= "return" [<expression>] ";"

<break\_statement> ::= "break" ";"

<expression> ::= <term> {("+" | "-" | "\*" | "/") <term>}

<term> ::= <variable\_name> | <literal> | <method\_call> | <object\_creation>

<literal> ::= <numeric\_literal> | <char\_literal> | <boolean\_literal> | <string\_literal>

<numeric\_literal> ::= <digit> {"." <digit>} ["d"]

<char\_literal> ::= "'" <character> "'"

<boolean\_literal> ::= "true" | "false"

<method\_call> ::= <object\_name> "." <method\_name> "(" [<argument\_list>] ")"

<object\_name> ::= <variable\_name>

<argument\_list> ::= [<expression> {"," <expression>}]

<object\_creation> ::= <class\_name> <object\_name> "=" "new" <class\_name> "(" ")" ";"

<string\_literal> ::= "\"" {<any\_character>} "\""

<any\_character> ::= <letter> | <digit> | <special\_character>

<letter\_or\_digit> ::= <letter> | <digit>

<letter> ::= <uppercase\_letter> | <lowercase\_letter>

<uppercase\_letter> ::= "A" | "B" | ... | "Z"

<lowercase\_letter> ::= "a" | "b" | ... | "z"

<digit> ::= "0" | "1" | ... | "9"

<special\_character> ::= "\_" | "+" | "-" | "\*" | "/" | "." | "," | ";" | ":" | "(" | ")" | "{" | "}" | "[" | "]" | "<" | ">" | "=" | "!" | "&" | "|" | "'" | "\"" | " " | "\n"

**Δοκιμή Compiler**

Στο αρχείο εισόδου **sample.txt** προσθέτουμε τις εξής δηλώσεις μεταβλητών.

int var1 = 12;

var4 = var7 + 1;

public String my\_str = “H3ll0”;

int x = 15, y = 16, z = 17;

double x, y, z;

A black screen with white text

Description automatically generated

Εκτελούμε πάλι τα Flex, Bison

A black screen with white text

Description automatically generated

Και τρέχουμε και τον parser :  
  
A black background with white text

Description automatically generated

Εάν τώρα αλλάξουμε μια από τις καινούργιες μεταβλητές που προσθέσαμε θα εξασφαλίσουμε τη σωστή λειτουργία του compiler.

A screen shot of a computer program

Description automatically generated

Όπως φαίνεται, αφαιρούμε την τιμή ανάθεσης από τη μεταβλητή var1.

Τρέχουμε πάλι τον parser :

A black background with white text

Description automatically generated

Φαίνεται να αναγνωρίζει με επιτυχία το λάθος ο compiler μας.

**Ερώτημα 4**

Στο αρχείο Bison εισήχθησαν προσαρμογές για να καταστεί δυνατός ο χειρισμός σφαλμάτων σε διάφορα στάδια της παραγωγής της γραμματικής. Κανόνες σφάλματος ενσωματώθηκαν στη γραμματική, όπως στον κανόνα `class\_list`, για τη διαχείριση συντακτικών σφαλμάτων που παρουσιάζονται κατά την ανάλυση. Αυτοί οι κανόνες σφάλματος, που ενεργοποιούνται όταν εμφανίζονται συντακτικά σφάλματα, επικαλούνται τη συνάρτηση `yyerror` για να αναφέρουν το πρόβλημα και στη συνέχεια προτρέπουν τον αναλυτή να συνεχίσει (`yyerrok`), ενώ παράλληλα καθαρίζουν την είσοδο (`yyclearin`). Παρόμοιοι μηχανισμοί ανάκτησης σφαλμάτων εφαρμόστηκαν σε όλη τη γραμματική, εξασφαλίζοντας την ικανότητα του αναλυτή να ανακάμπτει από συντακτικά σφάλματα χωρίς να κολλάει σε βρόχο.

Αντίθετα, στο αρχείο Flex, δεν απαιτήθηκαν αλλαγές για τον χειρισμό των σφαλμάτων του αναλυτή, καθώς ο ρόλος του περιλαμβάνει κυρίως τη μαρκαρίωση της εισόδου. Αντ' αυτού, η εστίαση ήταν αποκλειστικά στη βελτίωση της ανάκτησης σφαλμάτων στο αρχείο Bison. Αυτές οι τροποποιήσεις διευκολύνουν την ικανότητα του αναλυτή να διαχειρίζεται αποτελεσματικά πολλαπλά συντακτικά σφάλματα εντός της εισόδου, εξασφαλίζοντας την ολοκληρωμένη αναφορά σφαλμάτων, διατηρώντας παράλληλα τη συνέχεια της διαδικασίας ανάλυσης μέχρι την ολοκλήρωση του αρχείου εισόδου.

Ο ανανεωμένος κώδικας Bison :   
  
%{

/\* Το αρχείο Bison  -> parser.y

Στο πρώτο τμήμα του ορίζεται ο κώδικας C\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

extern FILE \*yyin; // Εξωτερική δήλωση για το αρχείο εισόδου

extern int yylex(); // Εξωτερική δήλωση για τη συνάρτηση λεκτικής ανάλυσης

extern char \*yytext; // Εξωτερική δήλωση για τη μεταβλητή yytext

extern int yylineno; // Εξωτερική δήλωση για τη μεταβλητή yylineno

int error\_count = 0; // Μεταβλητή για την καταγραφή του αριθμού των σφαλμάτων

int flag\_err\_type = 0; // Μεταβλητή για τον έλεγχο του είδους του σφάλματος

int scope = 0; // Μεταβλητή για το επίπεδο εμβέλειας των μεταβλητών

void yyerror(const char \*message); // Πρότυπο για τη συνάρτηση διαχείρισης σφαλμάτων

%}

%token PUBLIC PRIVATE VOID INT CHAR DOUBLE BOOLEAN STRING CLASS NEW RETURN IF ELSE WHILE DO FOR SWITCH CASE DEFAULT BREAK TRUE FALSE OUTPRINT OUTPRINTLN

%token EQ\_OP COLON NE\_OP GE\_OP LE\_OP AND\_OP OR\_OP EROTIMATIKO COMMA EQUALS GREATER\_THAN LESS\_THAN PLUS MINUS STAR SLASH

%token T\_VAR T\_INTEGER T\_DOUBLE T\_STR T\_INTEGER\_STRING LEFT\_PAR RIGHT\_PAR LEFT\_BRC RIGHT\_BRC QT

%start program

// Grammar rules

%%

program:

    class\_list

    ;

class\_list:

    class\_list class\_decl

    | class\_decl

    | error { yyerror("Syntax error in class list"); yyerrok; yyclearin; }

    ;

class\_decl:

    PUBLIC CLASS identifier RIGHT\_BRC var\_decls method\_decls LEFT\_BRC

    | error { yyerror("Syntax error in class declaration"); yyerrok; yyclearin; }

    ;

method\_decls:

    method\_decls method\_decl

    | method\_decl

    | %empty

    ;

method\_decl:

    modifier type identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    | identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    | modifier VOID identifier RIGHT\_PAR param\_list LEFT\_PAR RIGHT\_BRC var\_decls opt\_statements LEFT\_BRC

    | error { yyerror("Syntax error in method declaration"); yyerrok; yyclearin; }

    ;

var\_decls:

    var\_decls var\_decl

    | var\_decl

    | %empty

    ;

var\_decl:

    type var\_decl\_list EROTIMATIKO

    | modifier type var\_decl\_list EROTIMATIKO

    | error { yyerror("Syntax error in variable declaration"); yyerrok; yyclearin; while (yychar != EROTIMATIKO && yychar != LEFT\_BRC && yychar != RIGHT\_BRC) yylex(); }

    ;

var\_decl\_list:

    var\_decl\_list COMMA var\_assign

    | var\_assign

    | error { yyerror("Syntax error in variable declaration list"); yyerrok; yyclearin; }

    ;

var\_assign:

    identifier EQUALS expression

    | identifier

    | error { yyerror("Syntax error in variable assignment"); yyerrok; yyclearin; }

    ;

modifier:

    PUBLIC

    | PRIVATE

    ;

type:

    INT

    | CHAR

    | DOUBLE

    | BOOLEAN

    | STRING

    ;

param\_list:

    param\_list COMMA param

    | param

    | %empty

    ;

param:

    type identifier

    | error { yyerror("Syntax error in parameter"); yyerrok; yyclearin; }

    ;

opt\_statements:

    statement\_list

    | %empty /\* empty \*/

    ;

statement\_list:

    statement\_list statement

    | statement

    | error { yyerror("Syntax error in statement list"); yyerrok; yyclearin; }

    ;

statement:

    var\_decl

    | assignment

    | if\_statement

    | while\_statement

    | do\_while\_statement

    | for\_statement

    | switch\_statement

    | out\_print

    | return\_statement

    | break\_statement

    | error { yyerror("Syntax error in statement"); yyerrok; yyclearin; while (yychar != EROTIMATIKO && yychar != LEFT\_BRC && yychar != RIGHT\_BRC) yylex(); }

    ;

assignment:

    identifier EQUALS expression EROTIMATIKO |

    identifier EQUALS expression

    | error { yyerror("Syntax error in assignment"); yyerrok; yyclearin; while (yychar != EROTIMATIKO && yychar != LEFT\_BRC && yychar != RIGHT\_BRC) yylex(); }

    ;

if\_statement:

    IF RIGHT\_PAR expression LEFT\_PAR RIGHT\_BRC opt\_statements LEFT\_BRC opt\_else

    | error { yyerror("Syntax error in if statement"); yyerrok; yyclearin; }

    ;

opt\_else:

    ELSE RIGHT\_BRC opt\_statements LEFT\_BRC

    | %empty /\* empty \*/

    ;

while\_statement:

    WHILE RIGHT\_PAR expression LEFT\_PAR RIGHT\_PAR opt\_statements LEFT\_PAR

    | error { yyerror("Syntax error in while statement"); yyerrok; yyclearin; }

    ;

do\_while\_statement:

    DO RIGHT\_BRC opt\_statements LEFT\_BRC WHILE RIGHT\_PAR expression LEFT\_PAR EROTIMATIKO

    | error { yyerror("Syntax error in do-while statement"); yyerrok; yyclearin; }

    ;

for\_statement:

    FOR RIGHT\_PAR type assignment expression EROTIMATIKO assignment LEFT\_PAR RIGHT\_BRC opt\_statements LEFT\_PAR

    | error { yyerror("Syntax error in for statement"); yyerrok; yyclearin; }

    ;

switch\_statement:

    SWITCH RIGHT\_PAR expression LEFT\_PAR RIGHT\_BRC case\_list opt\_default\_case LEFT\_BRC

    | error { yyerror("Syntax error in switch statement"); yyerrok; yyclearin; }

    ;

case\_list:

    case\_list case\_clause

    | case\_clause

    | error { yyerror("Syntax error in case list"); yyerrok; yyclearin; }

    ;

case\_clause:

    CASE expression COLON opt\_statements

    | error { yyerror("Syntax error in case clause"); yyerrok; yyclearin; }

    ;

opt\_default\_case:

    DEFAULT COLON opt\_statements

    | %empty /\* empty \*/

    ;

out\_print:

    OUTPRINT RIGHT\_PAR QT var\_list QT LEFT\_PAR EROTIMATIKO

    | OUTPRINTLN RIGHT\_PAR QT var\_list QT LEFT\_PAR EROTIMATIKO

    | error { yyerror("Syntax error in out print statement"); yyerrok; yyclearin; while (yychar != EROTIMATIKO && yychar != LEFT\_BRC && yychar != RIGHT\_BRC) yylex(); }

    ;

var\_list:

    var\_list T\_VAR

    | T\_VAR

    ;

return\_statement:

    RETURN expression EROTIMATIKO

    | RETURN EROTIMATIKO

    | error { yyerror("Syntax error in return statement"); yyerrok; yyclearin; }

    ;

break\_statement:

    BREAK EROTIMATIKO

    | error { yyerror("Syntax error in break statement"); yyerrok; yyclearin; }

    ;

expression:

    expression PLUS expression

    | expression MINUS expression

    | expression STAR expression

    | expression SLASH expression

    | expression GREATER\_THAN expression

    | expression LESS\_THAN expression

    | expression GE\_OP expression

    | expression LE\_OP expression

    | expression EQ\_OP expression

    | expression NE\_OP expression

    | expression AND\_OP expression

    | expression OR\_OP expression

    | RIGHT\_PAR expression LEFT\_PAR

    | identifier

    | T\_INTEGER

    | T\_DOUBLE

    | T\_STR

    | error { yyerror("Syntax error in expression"); yyerrok; yyclearin; while (yychar != EROTIMATIKO && yychar != LEFT\_BRC && yychar != RIGHT\_BRC) yylex(); }

    ;

identifier:

    T\_VAR

    | error { yyerror("Syntax error in identifier"); yyerrok; yyclearin; }

    ;

%%

int main(int argc, char \*argv[]){

    /\* If statement για να ελέγξει το αρχείο εισόδου \*/

    if(argc > 1){

        yyin = fopen(argv[1], "r");

        if (yyin == NULL){

            perror ("Σφάλμα ανοίγοντας το αρχείο"); return -1;

        }

    }

    /\* Καλείται η συνάρτηση yyparse που επιστρέφει 0 αν το input einai έγκυρο

         και 1 εάν υπάρχει συντακικό λάθος\*/

    yyparse();

    /\* Κλείσιμο αρχείου εισόδου\*/

    fclose(yyin);

    if(error\_count > 0){

        printf("Η συντακτική ανάλυση απέτυχε! Βρέθηκαν %d σφάλματα\n", error\_count);

    }else{

        printf("Η συντακτική ανάλυση ολοκληρώθηκε με επιτυχία! \nΔεν βρέθηκαν σφάλματα!\n");

    }

    return 0;

}

void yyerror(const char \*message)

{

    /\* Μετρητής για αριθμό σφαλμάτων \*/

    error\_count++;

    printf("-> Σφάλμα στη γραμμή %d που προέκυψε από %s : %s\n", yylineno, yytext, message);

    flag\_err\_type = 0; // Reset flag\_err\_type

}

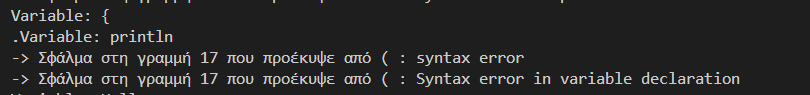
**Δοκιμή του Compiler**

**A screenshot of a computer program

Description automatically generated**

Στο αρχείο εισόδου μας sample.txt κάνουμε 3 αλλαγές για να δούμε εάν δουλεύει η τροποποίηση που κάναμε προκειμένου να αναγνωρίζει πολλαπλά λάθη το compiler.

Εκτελόυμε τον parser   
  

A screenshot of a computer screen

Description automatically generated

Ο compiler φαίνεται να αναγνωρίζει πολλαπλά σφάλματα χωρίς να σταματάει. Όμως παρατηρούμε οτι εντοπίζει παραπάνω λάθη από τα λογικά. Επίσης, παρατηρείται ότι ο parser για να αναλύσει όλα το αρχείο παίρνει πολύ χρόνο.