**Μέλη ομάδας:**

**1**

**2**

**3**

**4**

Contents

[Ερώτημα 1 2](#_Toc169143765)

[Flex 2](#_Toc169143766)

[BNF 5](#_Toc169143767)

[Parser 10](#_Toc169143768)

[Screenshots και δοκιμές πρώτου ερωτήματος 23](#_Toc169143769)

[Ερώτημα 2 26](#_Toc169143770)

[BNF 26](#_Toc169143771)

[Parser 32](#_Toc169143772)

[Screenshots και δοκιμές πρώτου ερωτήματος 46](#_Toc169143773)

[Ερώτημα 3 48](#_Toc169143774)

[Flex 48](#_Toc169143775)

[Parser 51](#_Toc169143776)

[Screenshots και δοκιμές πρώτου ερωτήματος 71](#_Toc169143777)

[Ερώτημα 4 72](#_Toc169143778)

[Parser 72](#_Toc169143779)

[Screenshots και δοκιμές πρώτου ερωτήματος 88](#_Toc169143780)

# Ερώτημα 1

## Flex

Το αρχικό μας flex.

%{

#include "parser.tab.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void yyerror(const char \*s);

%}

%option yylineno

%%

"public" { return PUBLIC; }

"private" { return PRIVATE; }

"class" { return CLASS; }

"int" { return INT; }

"char" { return CHAR; }

"double" { return DOUBLE; }

"boolean" { return BOOLEAN; }

"String" { return STRING; }

"void" { return VOID; }

"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 BOOLEAN\_LITERAL; }

"false" { return BOOLEAN\_LITERAL; }

"out.print" { return OUT\_PRINT; }

"then" { return THEN; }

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

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

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

"(" { return LPAREN; }

")" { return RPAREN; }

"{" { return LBRACE; }

"}" { return RBRACE; }

";" { return SEMICOLON; }

"," { return COMMA; }

"=" { return ASSIGN; }

"==" { return EQ; }

"!=" { return NEQ; }

"<" { return LT; }

">" { return GT; }

"<=" { return LE; }

">=" { return GE; }

"&&" { return AND; }

"||" { return OR; }

"+" { return PLUS; }

"-" { return MINUS; }

"\*" { return MUL; }

"/" { return DIV; }

"." { return DOT; }

":" { return COLON; }

\"([^\\\"]|\\.)\*\" { yylval.str = strdup(yytext); return STRING\_LITERAL; }

\'([^\\\']|\\.)\' { yylval.str = strdup(yytext); return CHAR\_LITERAL; }

[0-9]+(\.[0-9]+)?d? { yylval.num = atof(yytext); return NUMBER; }

[A-Z][A-Za-z0-9\_]\* { yylval.str = strdup(yytext); return CAPITALIZED\_IDENTIFIER; }

[a-zA-Z\_][A-Za-z0-9\_]\* { yylval.str = strdup(yytext); return IDENTIFIER; }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

## BNF

**<program> ::= <class\_declaration\_list>**

Ορίζει το σημείο εκκίνησης του προγράμματος, το οποίο αποτελείται από μία λίστα δηλώσεων κλάσεων.

**<class\_declaration\_list> ::= <class\_declaration> | <class\_declaration\_list> <class\_declaration>**

Ορίζει μία λίστα δηλώσεων κλάσεων. Μπορεί να είναι μία μεμονωμένη δήλωση κλάσης ή πολλές δηλώσεις κλάσεων.

**<class\_declaration> ::= "public" "class" <capitalized\_identifier> "{" <class\_body> "}"**

**| "class" <capitalized\_identifier> "{" <class\_body> "}"**

Ορίζει τη δήλωση μιας κλάσης, η οποία μπορεί να είναι δημόσια (public) ή χωρίς ρητό προσδιορισμό ορατότητας. Περιέχει το σώμα της κλάσης (class\_body).

**<class\_body> ::= <variable\_declaration\_list> <method\_declaration\_list>**

Ορίζει το σώμα της κλάσης, το οποίο αποτελείται από λίστες δηλώσεων μεταβλητών και μεθόδων.

**<variable\_declaration\_list> ::= <variable\_declaration>**

**| <variable\_declaration\_list> <variable\_declaration>**

**| /\* empty \*/**

Ορίζει μία λίστα δηλώσεων μεταβλητών, η οποία μπορεί να είναι άδεια.

**<variable\_declaration> ::= <type> <identifier> ";"**

**| <type> <identifier> "=" <expression> ";"**

**| <modifier> <type> <identifier> ";"**

**| <modifier> <type> <identifier> "=" <expression> ";"**

Ορίζει τη δήλωση μιας μεταβλητής με ή χωρίς αρχικοποίηση, και με ή χωρίς προσδιορισμό ορατότητας (public ή private).

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

Ορίζει τους προσδιορισμούς ορατότητας που μπορούν να χρησιμοποιηθούν σε μεταβλητές και μεθόδους.

**<method\_declaration\_list> ::= <method\_declaration>**

**| <method\_declaration\_list> <method\_declaration>**

**| /\* empty \*/**

Ορίζει μία λίστα δηλώσεων μεθόδων, η οποία μπορεί να είναι άδεια.

**<method\_declaration> ::= <modifier> <type> <identifier> "(" <parameter\_list> ")" "{" <method\_body> "}"**

**| <type> <identifier> "(" <parameter\_list> ")" "{" <method\_body> "}"**

Ορίζει τη δήλωση μιας μεθόδου, η οποία μπορεί να έχει ή να μην έχει προσδιορισμό ορατότητας. Περιέχει τις παραμέτρους της και το σώμα της μεθόδου.

**<parameter\_list> ::= <parameter>**

**| <parameter\_list> "," <parameter>**

**| /\* empty \*/**

Ορίζει μία λίστα παραμέτρων για μια μέθοδο, η οποία μπορεί να είναι άδεια.

**<parameter> ::= <type> <identifier>**

Ορίζει μία παράμετρο με τύπο και όνομα.

**<method\_body> ::= <statement\_list>**

Ορίζει το σώμα μιας μεθόδου, το οποίο αποτελείται από μία λίστα δηλώσεων.

**<statement\_list> ::= <statement>**

**| <statement\_list> <statement>**

**| /\* empty \*/**

Ορίζει μία λίστα δηλώσεων, η οποία μπορεί να είναι άδεια.

**<statement> ::= <assignment\_statement>**

**| <loop\_statement>**

**| <control\_statement>**

**| <print\_statement>**

**| <return\_statement>**

**| <break\_statement>**

**| <block\_statement>**

**| <switch\_statement>**

**| <variable\_declaration>**

Ορίζει τους διάφορους τύπους δηλώσεων που μπορούν να υπάρχουν σε ένα πρόγραμμα, όπως αναθέσεις, βρόχοι, έλεγχοι, εκτυπώσεις, επιστροφές, διακοπές, μπλοκ κώδικα, εναλλαγές και δηλώσεις μεταβλητών.

**<assignment\_statement> ::= <identifier> "=" <expression> "**

Ορίζει μία δήλωση ανάθεσης, όπου μια μεταβλητή λαμβάνει μια τιμή.

**<loop\_statement> ::= "do" "{" <statement\_list> "}" "while" "(" <condition> ")" ";"**

**| "for" "(" <for\_init> "" <condition\_opt> ";" <for\_increment> ")" "{" <statement\_list> "}"**

Ορίζει δηλώσεις βρόχων, όπως οι βρόχοι do\_while και for.

**<for\_init> ::= <variable\_declaration>**

**| <assignment\_statement>**

**| <expression>**

**| /\* empty \*/**

Ορίζει την αρχικοποίηση για έναν βρόχο for, η οποία μπορεί να είναι δήλωση μεταβλητής, ανάθεση, έκφραση ή άδεια.

**<for\_increment> ::= <assignment\_statement>**

**| <expression>**

Ορίζει την αύξηση για έναν βρόχο for, η οποία μπορεί να είναι ανάθεση ή έκφραση.

**<control\_statement> ::= "if" "(" <condition> ")" "{" <statement\_list> "}" %prec LOWER\_THAN\_ELSE**

**| "if" "(" <condition> ")" "{" <statement\_list> "}" "else" "{" <statement\_list> "}"**

Ορίζει δηλώσεις ελέγχου, όπως οι δηλώσεις if και if\_else.

**<switch\_statement> ::= "switch" "(" <expression> ")" "{" <case\_clauses> <default\_clause\_opt> "}"**

Ορίζει μία δήλωση εναλλαγής switch.

**<case\_clauses> ::= /\* empty \*/**

**| <case\_clauses> <case\_clause>**

Ορίζει τις περιπτώσεις case μέσα σε μία δήλωση switch, οι οποίες μπορούν να είναι άδειες.

**<case\_clause> ::= "case" <expression> ":" <statement\_list>**

Ορίζει μία μεμονωμένη περίπτωση case σε μία δήλωση switch.

**<default\_clause\_opt> ::= /\* empty \*/**

**| "default" ":" <statement\_list>**

Ορίζει μία προαιρετική περίπτωση default σε μία δήλωση switch.

**<print\_statement> ::= "out.print" "(" <string\_literal> <print\_argument\_list> ")" ";"**

Ορίζει μία δήλωση εκτύπωσης, η οποία χρησιμοποιεί την **out.print** μέθοδο για να εκτυπώσει τιμές.

**<print\_argument\_list> ::= /\* empty \*/**

**| "," <identifier>**

**| <print\_argument\_list> "," <identifier>**

Ορίζει μία λίστα επιχειρημάτων για την **out.print** μέθοδο, η οποία μπορεί να είναι άδεια.

**<return\_statement> ::= "return" <expression\_opt> ";"**

Ορίζει μία δήλωση επιστροφής, η οποία μπορεί να περιέχει μία προαιρετική έκφραση.

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

Ορίζει μία δήλωση διακοπής.

**<block\_statement> ::= "{" <statement\_list> "}"**

Ορίζει ένα μπλοκ κώδικα που περιέχει μία λίστα δηλώσεων.

**<expression\_opt> ::= /\* empty \*/**

**| <expression>**

Ορίζει μία προαιρετική έκφραση.

**<condition\_opt> ::= /\* empty \*/**

**| <condition>**

Ορίζει μία προαιρετική συνθήκη.

**<condition> ::= <expression> <relational\_operator> <expression>**

**| <expression>**

Ορίζει μία συνθήκη, η οποία μπορεί να είναι μία σύγκριση δύο εκφράσεων ή απλά μία έκφραση.

**<relational\_operator> ::= "<" | ">" | "==" | "!=" | "<=" | ">="**

Ορίζει τους λογικούς τελεστές σύγκρισης.

**<expression> ::= <literal>**

**| <identifier>**

**| <binary\_expression>**

**| <unary\_expression>**

**| <method\_call>**

**| <object\_creation>**

Ορίζει μία έκφραση, η οποία μπορεί να είναι μία κυριολεξία, ένας ταυτογράφος, μία δυαδική έκφραση, μία μοναδιαία έκφραση, μία κλήση μεθόδου ή μία δημιουργία αντικειμένου.

**<binary\_expression> ::= <expression> <binary\_operator> <expression>**

Ορίζει μία δυαδική έκφραση, η οποία περιλαμβάνει δύο εκφράσεις και έναν δυαδικό τελεστή.

**<unary\_expression> ::= <unary\_operator> <expression>**

Ορίζει μία μοναδιαία έκφραση, η οποία περιλαμβάνει έναν μοναδιαίο τελεστή και μία έκφραση.

**<method\_call> ::= <identifier> "." <identifier> "(" <argument\_list> ")"**

Ορίζει μία κλήση μεθόδου.

**<object\_creation> ::= "new" <identifier> "(" ")"**

Ορίζει τη δημιουργία ενός αντικειμένου.

**<argument\_list> ::= /\* empty \*/**

**| <expression>**

**| <argument\_list> "," <expression>**

Ορίζει μία λίστα επιχειρημάτων για μία κλήση μεθόδου, η οποία μπορεί να είναι άδεια.

**<literal> ::= <number>**

**| <char\_literal>**

**| <string\_literal>**

**| <boolean\_literal>**

Ορίζει μία κυριολεξία, η οποία μπορεί να είναι αριθμητική, χαρακτήρας, συμβολοσειρά ή λογική.

**<binary\_operator> ::= "+" | "-" | "\*" | "/" | "&&" | "||"**

Ορίζει τους δυαδικούς τελεστές για αριθμητικές και λογικές πράξεις.

**<unary\_operator> ::= "-" | "!"**

Ορίζει τους μοναδιαίους τελεστές για αρνητικές και λογικές πράξεις.

**<type> ::= "int"**

**| "char"**

**| "double"**

**| "boolean"**

**| "String"**

**| "void"**

**| <identifier>**

Ορίζει τους τύπους δεδομένων που μπορούν να χρησιμοποιηθούν.

**<identifier> ::= [a-zA-Z\_][a-zA-Z0-9\_]\***

**<capitalized\_identifier> ::= [A-Z][a-zA-Z0-9\_]\***

**<string\_literal> ::= "\""([^\\\"]|\\.)\*\""**

**<char\_literal> ::= "\'([^\\\']|\\.)\'"**

**<number> ::= [0-9]+(\.[0-9]+)?d?**

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

## Parser

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

extern FILE \*yyin;

extern int yylineno;

extern char \*yytext;

extern int yylex();

void yyerror(const char \*s);

// Error counter

int error\_count = 0;

%}

%union {

char \*str;

double num;

}

%token <str> IDENTIFIER

%token <str> CAPITALIZED\_IDENTIFIER

%token <str> STRING\_LITERAL

%token <str> CHAR\_LITERAL

%token <num> NUMBER

%token PUBLIC

%token PRIVATE

%token CLASS

%token INT

%token CHAR

%token DOUBLE

%token BOOLEAN

%token STRING

%token VOID

%token NEW

%token RETURN

%token IF

%token ELSE

%token WHILE

%token DO

%token FOR

%token SWITCH

%token CASE

%token DEFAULT

%token BREAK

%token BOOLEAN\_LITERAL

%token OUT\_PRINT

%token THEN

%token LPAREN

%token RPAREN

%token LBRACE

%token RBRACE

%token SEMICOLON

%token COMMA

%token ASSIGN

%token EQ

%token NEQ

%token LT

%token GT

%token LE

%token GE

%token AND

%token OR

%token PLUS

%token MINUS

%token MUL

%token DIV

%token DOT

%token COLON

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

%left OR

%left AND

%left EQ NEQ

%left LT GT

%left PLUS MINUS

%left MUL DIV

%right ASSIGN

%start program

%%

program

: class\_declaration\_list

;

class\_declaration\_list

: class\_declaration

| class\_declaration\_list class\_declaration

;

class\_declaration

: PUBLIC CLASS CAPITALIZED\_IDENTIFIER LBRACE class\_body RBRACE

| CLASS CAPITALIZED\_IDENTIFIER LBRACE class\_body RBRACE

;

class\_body

: variable\_declaration\_list method\_declaration\_list

;

variable\_declaration\_list

: variable\_declaration

| variable\_declaration\_list variable\_declaration

| /\* empty \*/

;

variable\_declaration

: type IDENTIFIER SEMICOLON

| type IDENTIFIER ASSIGN expression SEMICOLON

| modifier type IDENTIFIER SEMICOLON

| modifier type IDENTIFIER ASSIGN expression SEMICOLON

;

modifier

: PUBLIC

| PRIVATE

;

method\_declaration\_list

: method\_declaration

| method\_declaration\_list method\_declaration

| /\* empty \*/

;

method\_declaration

: modifier type IDENTIFIER LPAREN parameter\_list RPAREN LBRACE method\_body RBRACE

| type IDENTIFIER LPAREN parameter\_list RPAREN LBRACE method\_body RBRACE

;

parameter\_list

: parameter

| parameter\_list COMMA parameter

| /\* empty \*/

;

parameter

: type IDENTIFIER

;

method\_body

: statement\_list

;

statement\_list

: statement

| statement\_list statement

| /\* empty \*/

;

statement

: assignment\_statement

| loop\_statement

| control\_statement

| print\_statement

| return\_statement

| break\_statement

| block\_statement

| switch\_statement

| variable\_declaration

;

assignment\_statement

: IDENTIFIER ASSIGN expression

;

condition\_opt

: /\* empty \*/

| condition

;

loop\_statement

: DO LBRACE statement\_list RBRACE WHILE LPAREN condition RPAREN SEMICOLON

| FOR LPAREN for\_init condition\_opt SEMICOLON for\_increment RPAREN LBRACE statement\_list RBRACE

;

for\_init

: variable\_declaration

| expression

| /\* empty \*/

;

for\_increment

: assignment\_statement

| expression

| /\* empty \*/

;

control\_statement

: IF LPAREN condition RPAREN LBRACE statement\_list RBRACE %prec LOWER\_THAN\_ELSE

| IF LPAREN condition RPAREN LBRACE statement\_list RBRACE ELSE LBRACE statement\_list RBRACE

;

switch\_statement

: SWITCH LPAREN expression RPAREN LBRACE case\_clauses default\_clause\_opt RBRACE

;

case\_clauses

: /\* empty \*/

| case\_clauses case\_clause

;

case\_clause

: CASE expression COLON statement\_list

;

default\_clause\_opt

: /\* empty \*/

| DEFAULT COLON statement\_list

;

print\_statement

: OUT\_PRINT LPAREN STRING\_LITERAL print\_argument\_list RPAREN SEMICOLON

;

print\_argument\_list

: /\* empty \*/

| COMMA expression

| print\_argument\_list COMMA expression

;

return\_statement

: RETURN expression\_opt SEMICOLON

;

break\_statement

: BREAK SEMICOLON

;

block\_statement

: LBRACE statement\_list RBRACE

;

expression\_opt

: /\* empty \*/

| expression

;

condition

: expression relational\_operator expression

| expression

;

relational\_operator

: LT

| GT

| EQ

| NEQ

| LE

| GE

;

expression

: literal

| IDENTIFIER

| binary\_expression

| unary\_expression

| method\_call

| object\_creation

;

binary\_expression

: expression binary\_operator expression

;

unary\_expression

: unary\_operator expression

;

method\_call

: IDENTIFIER DOT IDENTIFIER LPAREN argument\_list RPAREN

;

object\_creation

: NEW IDENTIFIER LPAREN RPAREN

;

argument\_list

: /\* empty \*/

| expression

| argument\_list COMMA expression

;

literal

: NUMBER

| CHAR\_LITERAL

| STRING\_LITERAL

| BOOLEAN\_LITERAL

;

binary\_operator

: PLUS

| MINUS

| MUL

| DIV

| AND

| OR

;

unary\_operator

: MINUS

| "!"

;

type

: INT

| CHAR

| DOUBLE

| BOOLEAN

| STRING

| VOID

| IDENTIFIER

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s at line %d, token: %s\n", s, yylineno, yytext);

error\_count++;

}

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

if (argc > 1) {

FILE \*file = fopen(argv[1], "r");

if (!file) {

fprintf(stderr, "Could not open %s\n", argv[1]);

return 1;

}

yyin = file;

}

int result = yyparse();

if (error\_count > 0) {

fprintf(stderr, "Parsing failed with %d errors.\n", error\_count);

} else {

fprintf(stdout, "Parsing succeeded with no errors.\n");

}

return result;

}

## Screenshots και δοκιμές πρώτου ερωτήματος

**Ξεκινήσαμε δοκιμάζοντας ένα σωστό αρχείο με όνομα test\_file.txt.**

Το αρχείο:

A screenshot of a computer program

Description automatically generated

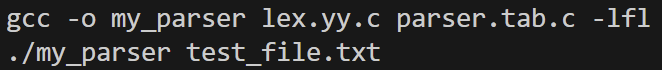
**Οι εντολές που χρησιμοποιήσαμε:**

A black background with white text

Description automatically generated

Η εντολή **flex lexer.l** μόλις εκτελεστεί παράγει τον λεκτικό αναλυτή σε γλώσσα C, και δημιουργεί το αρχείο **lex.yy.c**.

Η εντολή **bison -d parser.y** παράγει τον parser σε γλώσσα C, και δημιουργεί δύο αρχεία το **parser.tab.c** το οποίο είναι ο κώδικας C για τον parser και το **parser.tab.h** το οποίο περιέχει τους ορισμούς των tokens.



Η εντολή **gcc -o my\_parser lex.yy.c parser.tab.c -lfl** κάνει compile τα αρχεία που παράχθηκαν πιο πάνω. Δημιουργεί το **my\_parser** το οποιό χρησιμοποιούμε αργότερα. Το -lfl συνδέει τη βιβλιοθήκη του flex.

Η εντολή **./my\_parser test\_file.txt** τρέχει το **my\_parser** σε ένα αρχείο txt που έχουμε δημιουργήσει.

Ο κώδικας που δοκιμάσαμε τηρεί όλες τις προϋποθέσεις και τρέχει επιτυχώς.

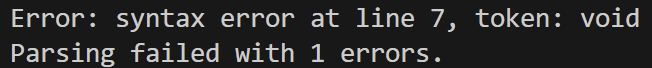


**Δοκιμάσαμε το ίδιο αρχείο απλά γράψαμε λάθος το public στη γραμμή 7.**

A black screen with white text

Description automatically generated

Στο τροποποιημένο αρχείο παρουσιάστηκε σφάλμα στη γραμμή 7.



# Ερώτημα 2

## BNF

Έχει προστεθεί επεξήγηση στα κομμάτια που άλλαξαν.

**<program> ::= <class\_declaration\_list>**

**<class\_declaration\_list> ::= <class\_declaration>**

**| <class\_declaration\_list> <class\_declaration>**

**<class\_declaration> ::= "public" "class" <CAPITALIZED\_IDENTIFIER> "{" <class\_body> "}"**

**| "class" <CAPITALIZED\_IDENTIFIER> "{" <class\_body> "}"**

**<class\_body> ::= <variable\_declaration\_list> <method\_declaration\_list>**

**<variable\_declaration\_list> ::= <variable\_declaration>**

**| <variable\_declaration\_list> <variable\_declaration>**

**| /\* empty \*/**

Ορίζει μία λίστα δηλώσεων μεταβλητών. Μπορεί να είναι μία μεμονωμένη δήλωση μεταβλητής, μία λίστα δηλώσεων μεταβλητών ή να είναι άδεια.

**<variable\_declaration> ::= <type> <variable\_declarators> ";"**

**| <modifier> <type> <variable\_declarators> ";"**

Ορίζει τη δήλωση μιας ή περισσότερων μεταβλητών. Μπορεί να περιλαμβάνει προσδιορισμό ορατότητας (modifier) και τύπο δεδομένων (type), και ακολουθείται από ένα ή περισσότερους δηλωτές μεταβλητών (variable\_declarators), ολοκληρώνοντας με έναν ελληνικό ερωτηματικό (;).

**<variable\_declarators> ::= <variable\_declarators> "," <variable\_declarator>**

**| <variable\_declarator>**

Ορίζει μία λίστα δηλωτών μεταβλητών, η οποία μπορεί να περιλαμβάνει περισσότερου1ς από έναν δηλωτές.

**<variable\_declarator> ::= <IDENTIFIER>**

**| <IDENTIFIER> "=" <expression>**

Ορίζει τη δήλωση ενός μεμονωμένου δηλωτή μεταβλητής. Μπορεί να είναι απλά ένα αναγνωριστικό (identifier) ή ένα αναγνωριστικό με αρχικοποίηση τιμής.

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

**<method\_declaration\_list> ::= <method\_declaration>**

**| <method\_declaration\_list> <method\_declaration>**

**| /\* empty \*/**

**<method\_declaration> ::= <modifier> <type> <IDENTIFIER> "(" <parameter\_list> ")" "{" <method\_body> "}"**

**| <type> <IDENTIFIER> "(" <parameter\_list> ")" "{" <method\_body> "}"**

**<parameter\_list> ::= <parameter>**

**| <parameter\_list> "," <parameter>**

**| /\* empty \*/**

**<parameter> ::= <type> <IDENTIFIER>**

**<method\_body> ::= <statement\_list>**

**<statement\_list> ::= <statement>**

**| <statement\_list> <statement>**

**| /\* empty \*/**

**<statement> ::= <assignment\_statement>**

**| <loop\_statement>**

**| <control\_statement>**

**| <print\_statement>**

**| <return\_statement>**

**| <break\_statement>**

**| <block\_statement>**

**| <switch\_statement>**

**| <variable\_declaration>**

**<assignment\_statement> ::= <IDENTIFIER> "=" <expression> ";"**

**<loop\_statement> ::= "do" "{" <statement\_list> "}" "while" "(" <condition> ")" ";"**

**| "for" "(" <for\_init> <condition\_opt> ";" <for\_increment> ")" "{" <statement\_list> "}"**

**<for\_init> ::= <variable\_declaration>**

**| <assignment\_statement>**

**| <expression>**

**| /\* empty \*/**

**<for\_increment> ::= <assignment\_statement>**

**| <expression>**

**<control\_statement> ::= "if" "(" <condition> ")" "{" <statement\_list> "}" %prec LOWER\_THAN\_ELSE**

**| "if" "(" <condition> ")" "{" <statement\_list> "}" "else" "{" <statement\_list> "}"**

**<switch\_statement> ::= "switch" "(" <expression> ")" "{" <case\_clauses> <default\_clause\_opt> "}"**

**<case\_clauses> ::= /\* empty \*/**

**| <case\_clauses> <case\_clause>**

**<case\_clause> ::= "case" <expression> ":" <statement\_list>**

**<default\_clause\_opt> ::= /\* empty \*/**

**| "default" ":" <statement\_list>**

**<print\_statement> ::= "out.print" "(" <STRING\_LITERAL> <print\_argument\_list> ")" ";"**

**<print\_argument\_list> ::= /\* empty \*/**

**| "," <expression>**

**| <print\_argument\_list> "," <expression>**

**<return\_statement> ::= "return" <expression\_opt> ";"**

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

**<block\_statement> ::= "{" <statement\_list> "}"**

**<expression\_opt> ::= /\* empty \*/**

**| <expression>**

**<condition\_opt> ::= /\* empty \*/**

**| <condition>**

**<condition> ::= <expression> <relational\_operator> <expression>**

**| <expression>**

**<relational\_operator> ::= "<" | ">" | "==" | "!=" | "<=" | ">="**

**<expression> ::= <literal>**

**| <IDENTIFIER>**

**| <binary\_expression>**

**| <unary\_expression>**

**| <method\_call>**

**| <object\_creation>**

**<binary\_expression> ::= <expression> <binary\_operator> <expression>**

**<unary\_expression> ::= <unary\_operator> <expression>**

**<method\_call> ::= <IDENTIFIER> "." <IDENTIFIER> "(" <argument\_list> ")"**

**<object\_creation> ::= "new" <IDENTIFIER> "(" ")"**

**<argument\_list> ::= /\* empty \*/**

**| <expression>**

**| <argument\_list> "," <expression>**

**<literal> ::= <NUMBER>**

**| <CHAR\_LITERAL>**

**| <STRING\_LITERAL>**

**| <BOOLEAN\_LITERAL>**

**<binary\_operator> ::= "+" | "-" | "\*" | "/" | "&&" | "||"**

**<unary\_operator> ::= "-" | "!"**

**<type> ::= "int"**

**| "char"**

**| "double"**

**| "boolean"**

**| "String"**

**| "void"**

**| <IDENTIFIER>**

**<identifier> ::= [a-zA-Z\_][a-zA-Z0-9\_]\***

**<capitalized\_identifier> ::= [A-Z][a-zA-Z0-9\_]\***

**<string\_literal> ::= "\""([^\\\"]|\\.)\*\""**

**<char\_literal> ::= "\'([^\\\']|\\.)\'"**

**<number> ::= [0-9]+(\.[0-9]+)?d?**

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

## Parser

Οι αλλαγές για το δεύτερο ερώτημα είναι με **bold** παρακάτω.

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

extern FILE \*yyin;

extern int yylineno;

extern char \*yytext;

extern int yylex();

void yyerror(const char \*s);

int error\_count = 0;

%}

%union {

char \*str;

double num;

}

%token <str> IDENTIFIER

%token <str> CAPITALIZED\_IDENTIFIER

%token <str> STRING\_LITERAL

%token <str> CHAR\_LITERAL

%token <num> NUMBER

%token PUBLIC

%token PRIVATE

%token CLASS

%token INT

%token CHAR

%token DOUBLE

%token BOOLEAN

%token STRING

%token VOID

%token NEW

%token RETURN

%token IF

%token ELSE

%token WHILE

%token DO

%token FOR

%token SWITCH

%token CASE

%token DEFAULT

%token BREAK

%token BOOLEAN\_LITERAL

%token OUT\_PRINT

%token THEN

%token LPAREN

%token RPAREN

%token LBRACE

%token RBRACE

%token SEMICOLON

%token COMMA

%token ASSIGN

%token EQ

%token NEQ

%token LT

%token GT

%token LE

%token GE

%token AND

%token OR

%token PLUS

%token MINUS

%token MUL

%token DIV

%token DOT

%token COLON

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

%left OR

%left AND

%left EQ NEQ

%left LT GT LE GE

%left PLUS MINUS

%left MUL DIV

%right ASSIGNvimeo

%start program

%%

program

: class\_declaration\_list

;

class\_declaration\_list

: class\_declaration

| class\_declaration\_list class\_declaration

;

class\_declaration

: PUBLIC CLASS CAPITALIZED\_IDENTIFIER LBRACE class\_body RBRACE

| CLASS CAPITALIZED\_IDENTIFIER LBRACE class\_body RBRACE

;

class\_body

: variable\_declaration\_list method\_declaration\_list

;

**variable\_declaration\_list**

**: variable\_declaration**

**| variable\_declaration\_list variable\_declaration**

**| /\* empty \*/**

**;**

**variable\_declaration**

**: type variable\_declarators SEMICOLON**

**| modifier type variable\_declarators SEMICOLON**

**;**

**variable\_declarators**

**: variable\_declarators COMMA variable\_declarator**

**| variable\_declarator**

**;**

**variable\_declarator**

**: IDENTIFIER**

**| IDENTIFIER ASSIGN expression**

**;**

modifier

: PUBLIC

| PRIVATE

;

method\_declaration\_list

: method\_declaration

| method\_declaration\_list method\_declaration

| /\* empty \*/

;

method\_declaration

: modifier type IDENTIFIER LPAREN parameter\_list RPAREN LBRACE method\_body RBRACE

| type IDENTIFIER LPAREN parameter\_list RPAREN LBRACE method\_body RBRACE

;

parameter\_list

: parameter

| parameter\_list COMMA parameter

| /\* empty \*/

;

parameter

: type IDENTIFIER

;

method\_body

: statement\_list

;

statement\_list

: statement

| statement\_list statement

| /\* empty \*/

;

statement

: assignment\_statement

| loop\_statement

| control\_statement

| print\_statement

| return\_statement

| break\_statement

| block\_statement

| switch\_statement

| variable\_declaration

;

assignment\_statement

: IDENTIFIER ASSIGN expression

;

loop\_statement

: DO LBRACE statement\_list RBRACE WHILE LPAREN condition RPAREN SEMICOLON

| FOR LPAREN for\_init condition\_opt SEMICOLON for\_increment RPAREN LBRACE statement\_list RBRACE

;

for\_init

: variable\_declaration

| assignment\_statement

| expression

| /\* empty \*/

;

for\_increment

: assignment\_statement

| expression

;

control\_statement

: IF LPAREN condition RPAREN LBRACE statement\_list RBRACE %prec LOWER\_THAN\_ELSE

| IF LPAREN condition RPAREN LBRACE statement\_list RBRACE ELSE LBRACE statement\_list RBRACE

;

switch\_statement

: SWITCH LPAREN expression RPAREN LBRACE case\_clauses default\_clause\_opt RBRACE

;

case\_clauses

: /\* empty \*/

| case\_clauses case\_clause

;

case\_clause

: CASE expression COLON statement\_list

;

default\_clause\_opt

: /\* empty \*/

| DEFAULT COLON statement\_list

;

print\_statement

: OUT\_PRINT LPAREN STRING\_LITERAL print\_argument\_list RPAREN SEMICOLON

;

print\_argument\_list

: /\* empty \*/

| COMMA expression

| print\_argument\_list COMMA expression

;

return\_statement

: RETURN expression\_opt SEMICOLON

;

break\_statement

: BREAK SEMICOLON

;

block\_statement

: LBRACE statement\_list RBRACE

;

expression\_opt

: /\* empty \*/

| expression

;

condition\_opt

: /\* empty \*/

| condition

;

condition

: expression relational\_operator expression

| expression

;

relational\_operator

: LT

| GT

| EQ

| NEQ

| LE

| GE

;

expression

: literal

| IDENTIFIER

| binary\_expression

| unary\_expression

| method\_call

| object\_creation

;

binary\_expression

: expression binary\_operator expression

;

unary\_expression

: unary\_operator expression

;

method\_call

: IDENTIFIER DOT IDENTIFIER LPAREN argument\_list RPAREN

;

object\_creation

: NEW IDENTIFIER LPAREN RPAREN

;

argument\_list

: /\* empty \*/

| expression

| argument\_list COMMA expression

;

literal

: NUMBER

| CHAR\_LITERAL

| STRING\_LITERAL

| BOOLEAN\_LITERAL

;

binary\_operator

: PLUS

| MINUS

| MUL

| DIV

| AND

| OR

;

unary\_operator

: MINUS

| "!"

;

type

: INT

| CHAR

| DOUBLE

| BOOLEAN

| STRING

| VOID

| IDENTIFIER

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s at line %d, token: %s\n", s, yylineno, yytext);

error\_count++;

}

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

if (argc > 1) {

FILE \*file = fopen(argv[1], "r");

if (!file) {

fprintf(stderr, "Could not open %s\n", argv[1]);

return 1;

}

yyin = file;

}

int result = yyparse();

if (error\_count > 0) {

fprintf(stderr, "Parsing failed with %d errors.\n", error\_count);

} else {

fprintf(stdout, "Parsing succeeded with no errors.\n");

}

return result;

}

## Screenshots και δοκιμές πρώτου ερωτήματος

**Ξεκινήσαμε δοκιμάζοντας ένα σωστό αρχείο με όνομα test\_file1.txt.**

A screenshot of a computer program

Description automatically generated

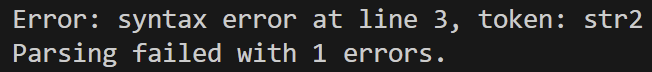


**Δοκιμάσαμε το ίδιο αρχείο αλλά δεν βάλαμε , πριν τη δήλωση του str2.**

A computer screen shot of a black screen

Description automatically generated

Στο τροποποιημένο αρχείο παρουσιάστηκε σφάλμα στη γραμμή 3.



# Ερώτημα 3

Στο τρίτο ερώτημα τροποποιήσαμε τον parser αρκετά αλλά προέκυψαν κάποια σφάλματα.

## Flex

Κάναμε κάποιες τροποποιήσεις και στο αρχείο flexer.l.

%{

#include "parser.tab.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

**extern void enter\_scope();**

**extern void exit\_scope();**

**extern void add\_symbol(char \*name, int is\_method);**

**extern int check\_symbol(char \*name, int is\_method);**

**extern void yyerror\_with\_expected(const char \*s, const char \*expected);**

%}

%option yylineno

%%

"public" { return PUBLIC; }

"private" { return PRIVATE; }

"class" { return CLASS; }

"int" { return INT; }

"char" { return CHAR; }

"double" { return DOUBLE; }

"boolean" { return BOOLEAN; }

"String" { return STRING; }

"void" { return VOID; }

"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 BOOLEAN\_LITERAL; }

"false" { return BOOLEAN\_LITERAL; }

"out.print" { return OUT\_PRINT; }

"then" { return THEN; }

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

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

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

"(" { return LPAREN; }

")" { return RPAREN; }

"{" { return LBRACE; }

"}" { return RBRACE; }

";" { return SEMICOLON; }

"," { return COMMA; }

"=" { return ASSIGN; }

"==" { return EQ; }

"!=" { return NEQ; }

"<" { return LT; }

">" { return GT; }

"<=" { return LE; }

">=" { return GE; }

"&&" { return AND; }

"||" { return OR; }

"+" { return PLUS; }

"-" { return MINUS; }

"\*" { return MUL; }

"/" { return DIV; }

"." { return DOT; }

":" { return COLON; }

\"([^\\\"]|\\.)\*\" { yylval.str = strdup(yytext); return STRING\_LITERAL; }

\'([^\\\']|\\.)\' { yylval.str = strdup(yytext); return CHAR\_LITERAL; }

[0-9]+(\.[0-9]+)?d? { yylval.num = atof(yytext); return NUMBER; }

[A-Z][A-Za-z0-9\_]\* { yylval.str = strdup(yytext); return CAPITALIZED\_IDENTIFIER; }

[a-zA-Z\_][A-Za-z0-9\_]\* { yylval.str = strdup(yytext); return IDENTIFIER; }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

## Parser

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

extern FILE \*yyin;

extern int yylineno;

extern char \*yytext;

extern int yylex();

void yyerror(const char \*s);

int error\_count = 0;

**typedef struct symbol {**

**char \*name;**

**int is\_method;**

**struct symbol \*next;**

**} symbol\_t;**

**typedef struct scope {**

**symbol\_t \*symbols;**

**struct scope \*next;**

**} scope\_t;**

**scope\_t \*current\_scope = NULL;**

**void enter\_scope() {**

**scope\_t \*new\_scope = (scope\_t \*)malloc(sizeof(scope\_t));**

**new\_scope->symbols = NULL;**

**new\_scope->next = current\_scope;**

**current\_scope = new\_scope;**

**printf("Entering new scope\n");**

**}**

**void exit\_scope() {**

**if (current\_scope == NULL) return; // Safety check**

**scope\_t \*old\_scope = current\_scope;**

**current\_scope = current\_scope->next;**

**symbol\_t \*symbols = old\_scope->symbols;**

**while (symbols) {**

**symbol\_t \*next = symbols->next;**

**free(symbols->name);**

**free(symbols);**

**symbols = next;**

**}**

**free(old\_scope);**

**printf("Exiting scope\n");**

**}**

**void add\_symbol(char \*name, int is\_method) {**

**if (current\_scope == NULL) return; // Safety check**

**symbol\_t \*new\_symbol = (symbol\_t \*)malloc(sizeof(symbol\_t));**

**new\_symbol->name = strdup(name);**

**new\_symbol->is\_method = is\_method;**

**new\_symbol->next = current\_scope->symbols;**

**current\_scope->symbols = new\_symbol;**

**printf("Adding symbol: %s (method: %d)\n", name, is\_method);**

**}**

**int check\_symbol(char \*name, int is\_method) {**

**if (current\_scope == NULL) return 0; // Safety check**

**scope\_t \*scope = current\_scope;**

**while (scope) {**

**symbol\_t \*symbols = scope->symbols;**

**while (symbols) {**

**if (strcmp(symbols->name, name) == 0 && symbols->is\_method == is\_method) {**

**return 1;**

**}**

**symbols = symbols->next;**

**}**

**scope = scope->next;**

**}**

**return 0;**

**}**

**void yyerror\_with\_expected(const char \*s, const char \*expected) {**

**fprintf(stderr, "Error: %s at line %d, token: %s. Expected: %s\n", s, yylineno, yytext, expected);**

**error\_count++;**

**}**

%}

%union {

char \*str;

double num;

}

%token <str> IDENTIFIER

%token <str> CAPITALIZED\_IDENTIFIER

%token <str> STRING\_LITERAL

%token <str> CHAR\_LITERAL

%token <num> NUMBER

%token PUBLIC

%token PRIVATE

%token CLASS

%token INT

%token CHAR

%token DOUBLE

%token BOOLEAN

%token STRING

%token VOID

%token NEW

%token RETURN

%token IF

%token ELSE

%token WHILE

%token DO

%token FOR

%token SWITCH

%token CASE

%token DEFAULT

%token BREAK

%token BOOLEAN\_LITERAL

%token OUT\_PRINT

%token THEN

%token LPAREN

%token RPAREN

%token LBRACE

%token RBRACE

%token SEMICOLON

%token COMMA

%token ASSIGN

%token EQ

%token NEQ

%token LT

%token GT

%token LE

%token GE

%token AND

%token OR

%token PLUS

%token MINUS

%token MUL

%token DIV

%token DOT

%token COLON

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

%left OR

%left AND

%left EQ NEQ

%left LT GT LE GE

%left PLUS MINUS

%left MUL DIV

%right ASSIGN

%start program

%%

program

: class\_declaration\_list

;

class\_declaration\_list

: class\_declaration

| class\_declaration\_list class\_declaration

;

class\_declaration

: PUBLIC CLASS CAPITALIZED\_IDENTIFIER LBRACE class\_body RBRACE

| CLASS CAPITALIZED\_IDENTIFIER LBRACE class\_body RBRACE

;

class\_body

: variable\_declaration\_list method\_declaration\_list

;

variable\_declaration\_list

: variable\_declaration

| variable\_declaration\_list variable\_declaration

| /\* empty \*/

;

variable\_declaration

: type IDENTIFIER SEMICOLON {

if (!check\_symbol($2, 0)) {

add\_symbol($2, 0);

} else {

yyerror\_with\_expected("Variable already declared", ";");

YYABORT;

}

}

| modifier type IDENTIFIER SEMICOLON {

if (!check\_symbol($3, 0)) {

add\_symbol($3, 0);

} else {

yyerror\_with\_expected("Variable already declared", ";");

YYABORT;

}

}

;

modifier

: PUBLIC

| PRIVATE

;

method\_declaration\_list

: method\_declaration

| method\_declaration\_list method\_declaration

| /\* empty \*/

;

method\_declaration

: modifier type IDENTIFIER LPAREN parameter\_list RPAREN LBRACE method\_body RBRACE {

if (!check\_symbol($3, 1)) {

add\_symbol($3, 1);

} else {

yyerror\_with\_expected("Method already declared", ")");

YYABORT;

}

}

| type IDENTIFIER LPAREN parameter\_list RPAREN LBRACE method\_body RBRACE {

if (!check\_symbol($2, 1)) {

add\_symbol($2, 1);

} else {

yyerror\_with\_expected("Method already declared", ")");

YYABORT;

}

}

;

parameter\_list

: parameter

| parameter\_list COMMA parameter

| /\* empty \*/

;

parameter

: type IDENTIFIER {

if (!check\_symbol($2, 0)) {

add\_symbol($2, 0);

} else {

yyerror\_with\_expected("Parameter already declared", ",");

YYABORT;

}

}

;

method\_body

: statement\_list

;

statement\_list

: statement

| statement\_list statement

| /\* empty \*/

;

statement

: assignment\_statement

| loop\_statement

| control\_statement

| print\_statement

| return\_statement

| break\_statement

| block\_statement

| switch\_statement

| variable\_declaration

;

assignment\_statement

: IDENTIFIER ASSIGN expression {

if (!check\_symbol($1, 0)) {

yyerror\_with\_expected("Variable not declared", "=");

YYABORT;

}

}

;

loop\_statement

: DO LBRACE statement\_list RBRACE WHILE LPAREN condition RPAREN SEMICOLON

| FOR LPAREN for\_init SEMICOLON condition\_opt SEMICOLON for\_increment RPAREN LBRACE statement\_list RBRACE

;

for\_init

: variable\_declaration

| assignment\_statement

| /\* empty \*/

;

for\_increment

: assignment\_statement

| expression

| /\* empty \*/

;

control\_statement

: IF LPAREN condition RPAREN LBRACE statement\_list RBRACE %prec LOWER\_THAN\_ELSE

| IF LPAREN condition RPAREN LBRACE statement\_list RBRACE ELSE LBRACE statement\_list RBRACE

;

switch\_statement

: SWITCH LPAREN expression RPAREN LBRACE case\_clauses default\_clause\_opt RBRACE

;

case\_clauses

: /\* empty \*/

| case\_clauses case\_clause

;

case\_clause

: CASE expression COLON statement\_list

;

default\_clause\_opt

: /\* empty \*/

| DEFAULT COLON statement\_list

;

print\_statement

: OUT\_PRINT LPAREN STRING\_LITERAL print\_argument\_list RPAREN SEMICOLON

;

print\_argument\_list

: /\* empty \*/

| COMMA expression

| print\_argument\_list COMMA expression

;

return\_statement

: RETURN expression\_opt SEMICOLON

;

break\_statement

: BREAK SEMICOLON

;

block\_statement

: LBRACE {

enter\_scope();

} statement\_list RBRACE {

exit\_scope();

}

;

expression\_opt

: /\* empty \*/

| expression

;

condition\_opt

: /\* empty \*/

| condition

;

condition

: expression relational\_operator expression

| expression

;

relational\_operator

: LT

| GT

| EQ

| NEQ

| LE

| GE

;

expression

: literal

| IDENTIFIER {

if (!check\_symbol($1, 0)) {

yyerror\_with\_expected("Variable not declared", $1);

YYABORT;

}

}

| binary\_expression

| unary\_expression

| method\_call

| object\_creation

;

binary\_expression

: expression binary\_operator expression

;

unary\_expression

: unary\_operator expression

;

method\_call

: IDENTIFIER LPAREN argument\_list RPAREN {

if (!check\_symbol($1, 1)) {

yyerror\_with\_expected("Method not declared", $1);

YYABORT;

}

}

| IDENTIFIER DOT IDENTIFIER LPAREN argument\_list RPAREN {

if (!check\_symbol($1, 0)) {

yyerror\_with\_expected("Variable not declared", $1);

YYABORT;

}

if (!check\_symbol($3, 1)) {

yyerror\_with\_expected("Method not declared", $3);

YYABORT;

}

}

;

object\_creation

: NEW IDENTIFIER LPAREN RPAREN {

if (!check\_symbol($2, 0)) {

yyerror\_with\_expected("Class not declared", $2);

YYABORT;

}

}

;

argument\_list

: /\* empty \*/

| expression

| argument\_list COMMA expression

;

literal

: NUMBER

| CHAR\_LITERAL

| STRING\_LITERAL

| BOOLEAN\_LITERAL

;

binary\_operator

: PLUS

| MINUS

| MUL

| DIV

| AND

| OR

;

unary\_operator

: MINUS

| "!"

;

type

: INT

| CHAR

| DOUBLE

| BOOLEAN

| STRING

| VOID

| IDENTIFIER

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s at line %d, token: %s\n", s, yylineno, yytext);

error\_count++;

}

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

// Initialize the global scope

enter\_scope();

if (argc > 1) {

FILE \*file = fopen(argv[1], "r");

if (!file) {

fprintf(stderr, "Could not open %s\n", argv[1]);

return 1;

}

yyin = file;

}

int result = yyparse();

exit\_scope();

if (error\_count > 0) {

fprintf(stderr, "Parsing failed with %d errors.\n", error\_count);

} else {

fprintf(stdout, "Parsing succeeded with no errors.\n");

}

return result;

}

## Screenshots και δοκιμές πρώτου ερωτήματος

**Ξεκινήσαμε δοκιμάζοντας ένα σωστό αρχείο με όνομα test\_file2.txt.**

A screenshot of a computer program

Description automatically generated

**Δοκιμάζοντας το πιο πάνω αρχείο προέκυψαν πολλά λάθη κατά τη δήλωση των μεταβλητών τα οποία διορθώσαμε τροποποιώντας τον parser, όμως στη συνέχεια προέκυψε κάτι το οποίο δεν καταφέραμε να διορθώσουμε.**

Παρουσιάστηκε σφάλμα στη γραμμή 11.

A screen shot of a computer program

Description automatically generated

# Ερώτημα 4

Προσπαθήσαμε να υλοποιήσουμε το ερώτημα αναγνωρίζοντας κάθε λάθος, αποθηκεύοντάς το και τέλος εκτυπώνοντας όλα τα λάθη μαζί. Προέκυψαν κάποια προβλήματα καθώς φαίνεται να εντοπίζεται το πρώτο λάθος και μετά είτε να σταματά είτε να εντοπίζει τα υπόλοιπα λάθη αλλά να μην εκτυπώνονται. Προσπαθήσαμε με κάποια print να κάνουμε debug.

## Parser

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "parser3.tab.h"

extern FILE \*yyin;

extern int yylineno;

extern char \*yytext;

extern int yylex();

void yyerror(const char \*s);

int error\_count = 0;

typedef struct Error {

int line;

char \*message;

struct Error \*next;

} Error;

Error \*error\_list = NULL;

void add\_error(int line, const char \*message) {

Error \*new\_error = (Error \*)malloc(sizeof(Error));

new\_error->line = line;

new\_error->message = strdup(message);

new\_error->next = error\_list;

error\_list = new\_error;

fprintf(stderr, "Stored error: %s at line %d\n", message, line);

}

void print\_errors() {

Error \*current = error\_list;

while (current != NULL) {

fprintf(stderr, "Printing error: %s at line %d\n", current->message, current->line); // Debugging message

fprintf(stderr, "Error: %s at line %d\n", current->message, current->line);

Error \*next = current->next;

free(current->message);

free(current);

current = next;

}

error\_list = NULL;

}

%}

%union {

char \*str;

double num;

}

%token <str> IDENTIFIER

%token <str> CAPITALIZED\_IDENTIFIER

%token <str> STRING\_LITERAL

%token <str> CHAR\_LITERAL

%token <num> NUMBER

%token PUBLIC

%token PRIVATE

%token CLASS

%token INT

%token CHAR

%token DOUBLE

%token BOOLEAN

%token STRING

%token VOID

%token NEW

%token RETURN

%token IF

%token ELSE

%token WHILE

%token DO

%token FOR

%token SWITCH

%token CASE

%token DEFAULT

%token BREAK

%token BOOLEAN\_LITERAL

%token OUT\_PRINT

%token THEN

%token LPAREN

%token RPAREN

%token LBRACE

%token RBRACE

%token SEMICOLON

%token COMMA

%token ASSIGN

%token EQ

%token NEQ

%token LT

%token GT

%token LE

%token GE

%token AND

%token OR

%token PLUS

%token MINUS

%token MUL

%token DIV

%token DOT

%token COLON

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

%left OR

%left AND

%left EQ NEQ

%left LT GT LE GE

%left PLUS MINUS

%left MUL DIV

%right ASSIGN

%start program

%%

program

: class\_declaration\_list

;

class\_declaration\_list

: class\_declaration

| class\_declaration\_list class\_declaration

;

class\_declaration

: PUBLIC CLASS CAPITALIZED\_IDENTIFIER LBRACE class\_body RBRACE

| CLASS CAPITALIZED\_IDENTIFIER LBRACE class\_body RBRACE

;

class\_body

: variable\_declaration\_list method\_declaration\_list

;

variable\_declaration\_list

: variable\_declaration

| variable\_declaration\_list variable\_declaration

| /\* empty \*/

;

variable\_declaration

: type variable\_declarators SEMICOLON

| modifier type variable\_declarators SEMICOLON

;

variable\_declarators

: variable\_declarators COMMA variable\_declarator

| variable\_declarator

;

variable\_declarator

: IDENTIFIER

| IDENTIFIER ASSIGN expression

;

modifier

: PUBLIC

| PRIVATE

;

method\_declaration\_list

: method\_declaration

| method\_declaration\_list method\_declaration

| /\* empty \*/

;

method\_declaration

: modifier type IDENTIFIER LPAREN parameter\_list RPAREN LBRACE method\_body RBRACE

| type IDENTIFIER LPAREN parameter\_list RPAREN LBRACE method\_body RBRACE

;

parameter\_list

: parameter

| parameter\_list COMMA parameter

| /\* empty \*/

;

parameter

: type IDENTIFIER

;

method\_body

: statement\_list

| error RBRACE { yyerrok; } // Skip to the end of method body

;

statement\_list

: statement

| statement\_list statement

| error SEMICOLON { yyerrok; } // Skip erroneous statements

;

statement

: assignment\_statement

| loop\_statement

| control\_statement

| print\_statement

| return\_statement

| break\_statement

| block\_statement

| switch\_statement

| variable\_declaration

;

assignment\_statement

: IDENTIFIER ASSIGN expression SEMCOLON

;

loop\_statement

: DO LBRACE statement\_list RBRACE WHILE LPAREN condition RPAREN SEMICOLON

| FOR LPAREN for\_init condition\_opt SEMICOLON for\_increment RPAREN LBRACE statement\_list RBRACE

;

for\_init

: variable\_declaration

| assignment\_statement

| expression

| /\* empty \*/

;

for\_increment

: assignment\_statement

| expression

;

control\_statement

: IF LPAREN condition RPAREN LBRACE statement\_list RBRACE %prec LOWER\_THAN\_ELSE

| IF LPAREN condition RPAREN LBRACE statement\_list RBRACE ELSE LBRACE statement\_list RBRACE

;

switch\_statement

: SWITCH LPAREN expression RPAREN LBRACE case\_clauses default\_clause\_opt RBRACE

;

case\_clauses

: /\* empty \*/

| case\_clauses case\_clause

;

case\_clause

: CASE expression COLON statement\_list

;

default\_clause\_opt

: /\* empty \*/

| DEFAULT COLON statement\_list

;

print\_statement

: OUT\_PRINT LPAREN STRING\_LITERAL print\_argument\_list RPAREN SEMICOLON

;

print\_argument\_list

: /\* empty \*/

| COMMA expression

| print\_argument\_list COMMA expression

;

return\_statement

: RETURN expression\_opt SEMICOLON

;

break\_statement

: BREAK SEMICOLON

;

block\_statement

: LBRACE statement\_list RBRACE

| LBRACE error RBRACE { yyerrok; } // Skip erroneous blocks

;

expression\_opt

: /\* empty \*/

| expression

;

condition\_opt

: /\* empty \*/

| condition

;

condition

: expression relational\_operator expression

| expression

;

relational\_operator

: LT

| GT

| EQ

| NEQ

| LE

| GE

;

expression

: literal

| IDENTIFIER

| binary\_expression

| unary\_expression

| method\_call

| object\_creation

;

binary\_expression

: expression binary\_operator expression

;

unary\_expression

: unary\_operator expression

;

method\_call

: IDENTIFIER DOT IDENTIFIER LPAREN argument\_list RPAREN

;

object\_creation

: NEW IDENTIFIER LPAREN RPAREN

;

argument\_list

: /\* empty \*/

| expression

| argument\_list COMMA expression

;

literal

: NUMBER

| CHAR\_LITERAL

| STRING\_LITERAL

| BOOLEAN\_LITERAL

;

binary\_operator

: PLUS

| MINUS

| MUL

| DIV

| AND

| OR

;

unary\_operator

: MINUS

| "!"

;

type

: INT

| CHAR

| DOUBLE

| BOOLEAN

| STRING

| VOID

| IDENTIFIER

;

%%

void yyerror(const char \*s) {

add\_error(yylineno, s);

error\_count++;

}

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

if (argc > 1) {

FILE \*file = fopen(argv[1], "r");

if (!file) {

fprintf(stderr, "Could not open %s\n", argv[1]);

return 1;

}

yyin = file;

}

int result = yyparse();

print\_errors(); // Print all errors at the end

if (error\_count > 0) {

fprintf(stderr, "Parsing failed with %d errors.\n", error\_count);

} else {

fprintf(stdout, "Parsing succeeded with no errors.\n");

}

return result;

}

## Screenshots και δοκιμές πρώτου ερωτήματος

**Ξεκινήσαμε δοκιμάζοντας ένα σωστό αρχείο με όνομα test\_file3.txt.**

A screenshot of a computer program

Description automatically generated

Παρουσιάστηκε σφάλμα στη γραμμή 3 αλλά δεν εμφανίστηκαν τα υπόλοιπα.

A black background with white text

Description automatically generated