Μέλη ομάδας

1

2

3

4

**Ερώτημα 1**

Α)

Παρακάτω παραθέτουμε το BNF.

<prog> ::= <class\_decl\_list>

<class\_decl\_list> ::= <class\_decl>

| <class\_decl\_list> <class\_decl>

<class\_decl> ::= 'public' 'class' <UPPER\_ID> '{' <class\_body> '}'

| 'class' <UPPER\_ID> '{' <class\_body> '}'

<class\_body> ::= <var\_decl\_list> <method\_decl\_list>

<var\_decl\_list> ::= <var\_decl>

| <var\_decl\_list> <var\_decl>

| /\* empty \*/

<var\_decl> ::= <ID> '=' <expr> ';'

| <data\_type> <ID> ';'

| <data\_type> <ID> '=' <expr> ';'

| <access\_mod> <data\_type> <ID> ';'

| <access\_mod> <data\_type> <ID> '=' <expr> ';'

<access\_mod> ::= 'public'

| 'private'

<method\_decl\_list> ::= <method\_decl>

| <method\_decl\_list> <method\_decl>

| /\* empty \*/

<method\_decl> ::= <access\_mod> <data\_type> <ID> '(' <param\_list> ')' '{' <method> '}'

| <data\_type> <ID> '(' <param\_list> ')' '{' <method> '}'

<param\_list> ::= <param>

| <param\_list> ',' <param>

| /\* empty \*/

<param> ::= <data\_type> <ID>

<method> ::= <stmt\_list>

<stmt\_list> ::= <stmt>

| <stmt\_list> <stmt>

| /\* empty \*/

<stmt> ::= <assign\_stmt>

| <loop\_stmt>

| <control\_stmt>

| <break\_stmt>

| <print\_stmt>

| <switch\_stmt>

| <return\_stmt>

| <block\_stmt>

| <var\_decl>

<assign\_stmt> ::= <ID> '=' <expr> ';'

<loop\_stmt> ::= 'do' '{' <stmt\_list> '}' 'while' '(' <condition> ')' ';'

| 'for' '(' <for\_1> <condition\_1> ';' <for\_2> ')' '{' <stmt\_list> '}'

<for\_1> ::= <var\_decl>

| <expr>

| /\* empty \*/

<for\_2> ::= <assign\_stmt>

| <expr>

| /\* empty \*/

<control\_stmt> ::= 'if' '(' <condition> ')' '{' <stmt\_list> '}' %prec LOWER\_THAN\_ELSE

| 'if' '(' <condition> ')' '{' <stmt\_list> '}' 'else' '{' <stmt\_list> '}'

<switch\_stmt> ::= 'switch' '(' <expr> ')' '{' <case\_clauses> <default\_clause\_opt> '}'

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

| <case\_clauses> <case\_clause>

<case\_clause> ::= 'case' <expr> ':' <stmt\_list>

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

| 'default' ':' <stmt\_list>

<print\_stmt> ::= 'out.print' '(' <STR\_LIT> <print\_arg\_list> ')' ';'

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

| ',' <expr>

| <print\_arg\_list> ',' <expr>

<return\_stmt> ::= 'return' <expr\_opt> ';'

<break\_stmt> ::= 'break' ';'

<block\_stmt> ::= '{' <stmt\_list> '}'

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

| <expr>

<condition> ::= <expr> <rel\_op> <expr>

| <expr>

<condition\_1> ::= /\* empty \*/

| <condition>

<rel\_op> ::= '<'

| '>'

| '=='

| '!='

| '<='

| '>='

<expr> ::= <literal>

| <ID>

| <binary\_expr>

| <unary\_expr>

| <method\_call>

| <obj\_create>

<binary\_expr> ::= <expr> <binary\_op> <expr>

<unary\_expr> ::= <unary\_op> <expr>

<method\_call> ::= <ID> '.' <ID> '(' <arg\_list> ')'

<obj\_create> ::= 'new' <ID> '(' ')'

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

| <expr>

| <arg\_list> ',' <expr>

<literal> ::= <NUM>

| <CHAR\_LIT>

| <STR\_LIT>

| <BOOL\_LIT>

<binary\_op> ::= '+'

| '-'

| '\*'

| '/'

| '&&'

| '||'

<unary\_op> ::= '-'

| '!'

<data\_type> ::= 'int'

| 'char'

| 'double'

| 'boolean'

| 'String'

| 'void'

| <ID>

B)

Παρακάτω παραθέτουμε τα flex και bison που δημιουργήσαμε για την υλοποίηση του προγράμματος μας.

Flex

%{

#include "bison.tab.h"

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

void yyerror(const char \*s);

%}

%option yylineno

%%

"int" { return INT; }

"char" { return CHAR; }

"double" { return DBL; }

"boolean" { return BOOL; }

"String" { return STR; }

"public" { return PUB; }

"private" { return PRIV; }

"class" { return CLS; }

"new" { return NEW; }

"return" { return RETURN; }

"void" { return VOID; }

"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 BOOL\_LIT; }

"false" { return BOOL\_LIT; }

"out.print" { return PRINT; }

"(" { return AR\_PAREN; }

")" { return DE\_PAREN; }

"{" { return AR\_AG; }

"}" { return DE\_AG; }

";" { return ERWT; }

"," { return KOM; }

"=" { return ISON; }

"+" { return SIN; }

"-" { return MEION ; }

"==" { return SINEP; }

"!=" { return INEK; }

"<" { return MIK; }

">" { return MEG; }

"<=" { return MIKIS; }

">=" { return MEGIS; }

"&&" { return AND; }

"||" { return OR; }

"\*" { return EPI; }

"/" { return DIA; }

"." { return TEL; }

":" { return ANKAT; }

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

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

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

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

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

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

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

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

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

Bison

%{

#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 = 0;

%}

%union {

char \*str;

double num;

}

%token <str> CHAR\_LIT

%token <str> STR\_LIT

%token <num> NUM

%token <str> UPPER\_ID

%token <str> ID

%token INT

%token CHAR

%token DBL

%token BOOL

%token STR

%token PUB

%token PRIV

%token CLS

%token NEW

%token RETURN

%token VOID

%token IF

%token ELSE

%token WHILE

%token DO

%token FOR

%token SWITCH

%token CASE

%token DEFAULT

%token BREAK

%token BOOL\_LIT

%token PRINT

%token AR\_PAREN

%token DE\_PAREN

%token AR\_AG

%token DE\_AG

%token ERWT

%token KOM

%token ISON

%token SIN

%token MEION

%token SINEP

%token INEK

%token MIK

%token MEG

%token MIKIS

%token MEGIS

%token AND

%token OR

%token EPI

%token DIA

%token TEL

%token ANKAT

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

%left OR

%left AND

%left SINEP INEK

%left MIK MEG

%left SIN MEION

%left EPI DIA

%right ISON

%start prog

%%

prog

: class\_decl\_list

;

class\_decl\_list

: class\_decl

| class\_decl\_list class\_decl

;

class\_decl

: PUB CLS UPPER\_ID AR\_AG class\_body DE\_AG

| CLS UPPER\_ID AR\_AG class\_body DE\_AG

;

class\_body

: var\_decl\_list method\_decl\_list

;

var\_decl\_list

: var\_decl

| var\_decl\_list var\_decl

| /\* empty \*/

;

var\_decl

: ID ISON expr ERWT

| data\_type ID ERWT

| data\_type ID ISON expr ERWT

| access\_mod data\_type ID ERWT

| access\_mod data\_type ID ISON expr ERWT

;

access\_mod

: PUB

| PRIV

;

method\_decl\_list

: method\_decl

| method\_decl\_list method\_decl

| /\* empty \*/

;

method\_decl

: access\_mod data\_type ID AR\_PAREN param\_list DE\_PAREN AR\_AG method DE\_AG

| data\_type ID AR\_PAREN param\_list DE\_PAREN AR\_AG method DE\_AG

;

param\_list

: param

| param\_list KOM param

| /\* empty \*/

;

param

: data\_type ID

;

method

: stmt\_list

;

stmt\_list

: stmt

| stmt\_list stmt

| /\* empty \*/

;

stmt

: assign\_stmt

| loop\_stmt

| control\_stmt

| break\_stmt

| print\_stmt

| switch\_stmt

| return\_stmt

| block\_stmt

| var\_decl

;

assign\_stmt

: ID ISON expr

;

loop\_stmt

: DO AR\_AG stmt\_list DE\_AG WHILE AR\_PAREN condition DE\_PAREN ERWT

| FOR AR\_PAREN for\_1 condition\_1 ERWT for\_2 DE\_PAREN AR\_AG stmt\_list DE\_AG

;

for\_1

: var\_decl

| expr

| /\* empty \*/

;

for\_2

: assign\_stmt

| expr

| /\* empty \*/

;

control\_stmt

: IF AR\_PAREN condition DE\_PAREN AR\_AG stmt\_list DE\_AG %prec LOWER\_THAN\_ELSE

| IF AR\_PAREN condition DE\_PAREN AR\_AG stmt\_list DE\_AG ELSE AR\_AG stmt\_list DE\_AG

;

switch\_stmt

: SWITCH AR\_PAREN expr DE\_PAREN AR\_AG case\_clauses default\_clause\_opt DE\_AG

;

case\_clauses

: /\* empty \*/

| case\_clauses case\_clause

;

case\_clause

: CASE expr ANKAT stmt\_list

;

default\_clause\_opt

: /\* empty \*/

| DEFAULT ANKAT stmt\_list

;

print\_stmt

: PRINT AR\_PAREN STR\_LIT print\_arg\_list DE\_PAREN ERWT

;

print\_arg\_list

: /\* empty \*/

| KOM expr

| print\_arg\_list KOM expr

;

return\_stmt

: RETURN expr\_opt ERWT

;

break\_stmt

: BREAK ERWT

;

block\_stmt

: AR\_AG stmt\_list DE\_AG

;

expr\_opt

: /\* empty \*/

| expr

;

condition

: expr rel\_op expr

| expr

;

condition\_1

: /\* empty \*/

| condition

;

binary\_expr

: expr binary\_op expr

;

unary\_expr

: unary\_op expr

;

expr

: literal

| ID

| binary\_expr

| unary\_expr

| method\_call

| obj\_create

;

rel\_op

: MIK

| MEG

| SINEP

| INEK

| MIKIS

| MEGIS

;

method\_call

: ID TEL ID AR\_PAREN arg\_list DE\_PAREN

;

obj\_create

: NEW ID AR\_PAREN DE\_PAREN

;

arg\_list

: /\* empty \*/

| expr

| arg\_list KOM expr

;

literal

: NUM

| CHAR\_LIT

| STR\_LIT

| BOOL\_LIT

;

binary\_op

: SIN

| MEION

| EPI

| DIA

| AND

| OR

;

unary\_op

: MEION

| "!"

;

data\_type

: INT

| CHAR

| DBL

| BOOL

| STR

| VOID

| ID

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Σφάλμα: %s στη γραμμή %d, τόκεν: %s\n", s, yylineno, yytext);

error++;

}

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

if (argc > 1) {

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

if (!file) {

fprintf(stderr, "Δεν μπόρεσα να ανοίξω το %s\n", argv[1]);

return 1;

}

yyin = file;

}

int result = yyparse();

if (error > 0) {

fprintf(stderr, "Η ανάλυση απέτυχε με %d σφάλματα.\n", error);

} else {

fprintf(stdout, "Η ανάλυση ολοκληρώθηκε με επιτυχία χωρίς σφάλματα.\n");

}

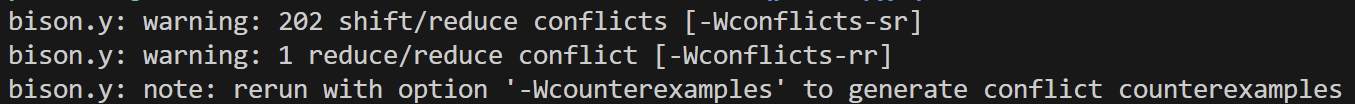
return result;

}

Παρακάτω παραθέτουμε screenshot και κώδικες από τις δοκιμές που κάναμε.

Οι εντολές που χρησιμοποιήσαμε:





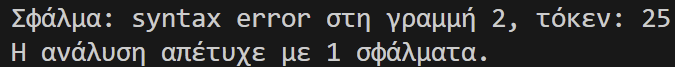
A screenshot of a computer program

Description automatically generatedΤο αρχείο που δοκιμάσαμε:



A screenshot of a computer program

Description automatically generatedΤο αρχείο που δοκιμάσαμε και έχει λάθος(δεν υπάρχει = στη γραμμή 2):



**Ερώτημα 2**

Παρακάτω παραθέτουμε τα τροποποιημένα BNF και bison.

BNF

<prog> ::= <class\_decl\_list>

<class\_decl\_list> ::= <class\_decl>

| <class\_decl\_list> <class\_decl>

<class\_decl> ::= PUB CLS UPPER\_ID AR\_AG <class\_body> DE\_AG

| CLS UPPER\_ID AR\_AG <class\_body> DE\_AG

<class\_body> ::= <var\_decl\_list> <method\_decl\_list>

<var\_decl\_list> ::= <var\_decl>

| <var\_decl\_list> <var\_decl>

| /\* empty \*/

<var\_decl> ::= <data\_type> <var\_declrs> ERWT

| <access\_mod> <data\_type> <var\_declrs> ERWT

<var\_declrs> ::= <var\_declr>

| <var\_declrs> KOM <var\_declr>

<var\_declr> ::= ID

| ID ISON <expr>

<access\_mod> ::= PUB

| PRIV

<method\_decl\_list> ::= <method\_decl>

| <method\_decl\_list> <method\_decl>

| /\* empty \*/

<method\_decl> ::= <access\_mod> <data\_type> ID AR\_PAREN <param\_list> DE\_PAREN AR\_AG <method> DE\_AG

| <data\_type> ID AR\_PAREN <param\_list> DE\_PAREN AR\_AG <method> DE\_AG

<param\_list> ::= <param>

| <param\_list> KOM <param>

| /\* empty \*/

<param> ::= <data\_type> ID

<method> ::= <stmt\_list>

<stmt\_list> ::= <stmt>

| <stmt\_list> <stmt>

| /\* empty \*/

<stmt> ::= <assign\_stmt>

| <loop\_stmt>

| <control\_stmt>

| <break\_stmt>

| <print\_stmt>

| <switch\_stmt>

| <return\_stmt>

| <block\_stmt>

| <var\_decl>

<assign\_stmt> ::= ID ISON <expr>

<loop\_stmt> ::= DO AR\_AG <stmt\_list> DE\_AG WHILE AR\_PAREN <condition> DE\_PAREN ERWT

| FOR AR\_PAREN <for\_1> ERWT <condition\_1> ERWT <for\_2> DE\_PAREN AR\_AG <stmt\_list> DE\_AG

<for\_1> ::= <var\_decl>

| <assign\_stmt>

| <expr>

| /\* empty \*/

<for\_2> ::= <assign\_stmt>

| <expr>

| /\* empty \*/

<control\_stmt> ::= IF AR\_PAREN <condition> DE\_PAREN AR\_AG <stmt\_list> DE\_AG %prec LOWER\_THAN\_ELSE

| IF AR\_PAREN <condition> DE\_PAREN AR\_AG <stmt\_list> DE\_AG ELSE AR\_AG <stmt\_list> DE\_AG

<switch\_stmt> ::= SWITCH AR\_PAREN <expr> DE\_PAREN AR\_AG <case\_clauses> <default\_clause\_opt> DE\_AG

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

| <case\_clauses> <case\_clause>

<case\_clause> ::= CASE <expr> ANKAT <stmt\_list>

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

| DEFAULT ANKAT <stmt\_list>

<print\_stmt> ::= PRINT AR\_PAREN STR\_LIT <print\_arg\_list> DE\_PAREN ERWT

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

| KOM <expr>

| <print\_arg\_list> KOM <expr>

<return\_stmt> ::= RETURN <expr\_opt> ERWT

<break\_stmt> ::= BREAK ERWT

<block\_stmt> ::= AR\_AG <stmt\_list> DE\_AG

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

| <expr>

<condition> ::= <expr> <rel\_op> <expr>

| <expr>

<condition\_1> ::= /\* empty \*/

| <condition>

<binary\_expr> ::= <expr> <binary\_op> <expr>

<unary\_expr> ::= <unary\_op> <expr>

<expr> ::= <literal>

| ID

| <binary\_expr>

| <unary\_expr>

| <method\_call>

| <obj\_create>

<rel\_op> ::= MIK

| MEG

| SINEP

| INEK

| MIKIS

| MEGIS

<method\_call> ::= ID TEL ID AR\_PAREN <arg\_list> DE\_PAREN

<obj\_create> ::= NEW ID AR\_PAREN DE\_PAREN

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

| <expr>

| <arg\_list> KOM <expr>

<literal> ::= NUM

| CHAR\_LIT

| STR\_LIT

| BOOL\_LIT

<binary\_op> ::= SIN

| MEION

| EPI

| DIA

| AND

| OR

<unary\_op> ::= MEION

| "!"

<data\_type> ::= INT

| CHAR

| DBL

| BOOL

| STR

| VOID

| ID

Bison

%{

#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 = 0;

%}

%union {

char \*str;

double num;

}

%token <str> CHAR\_LIT

%token <str> STR\_LIT

%token <num> NUM

%token <str> UPPER\_ID

%token <str> ID

%token INT

%token CHAR

%token DBL

%token BOOL

%token STR

%token PUB

%token PRIV

%token CLS

%token NEW

%token RETURN

%token VOID

%token IF

%token ELSE

%token WHILE

%token DO

%token FOR

%token SWITCH

%token CASE

%token DEFAULT

%token BREAK

%token BOOL\_LIT

%token PRINT

%token AR\_PAREN

%token DE\_PAREN

%token AR\_AG

%token DE\_AG

%token ERWT

%token KOM

%token ISON

%token SIN

%token MEION

%token SINEP

%token INEK

%token MIK

%token MEG

%token MIKIS

%token MEGIS

%token AND

%token OR

%token EPI

%token DIA

%token TEL

%token ANKAT

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

%left OR

%left AND

%left SINEP INEK

%left MIK MEG

%left SIN MEION

%left EPI DIA

%right ISON

%start prog

%%

prog

: class\_decl\_list

;

class\_decl\_list

: class\_decl

| class\_decl\_list class\_decl

;

class\_decl

: PUB CLS UPPER\_ID AR\_AG class\_body DE\_AG

| CLS UPPER\_ID AR\_AG class\_body DE\_AG

;

class\_body

: var\_decl\_list method\_decl\_list

;

var\_decl\_list

: var\_decl

| var\_decl\_list var\_decl

| /\* empty \*/

;

var\_decl

: data\_type var\_declrs ERWT

| access\_mod data\_type var\_declrs ERWT

;

var\_declrs

: var\_declr

| var\_declrs KOM var\_declr

;

var\_declr

: ID

| ID ISON expr

;

access\_mod

: PUB

| PRIV

;

method\_decl\_list

: method\_decl

| method\_decl\_list method\_decl

| /\* empty \*/

;

method\_decl

: access\_mod data\_type ID AR\_PAREN param\_list DE\_PAREN AR\_AG method DE\_AG

| data\_type ID AR\_PAREN param\_list DE\_PAREN AR\_AG method DE\_AG

;

param\_list

: param

| param\_list KOM param

| /\* empty \*/

;

param

: data\_type ID

;

method

: stmt\_list

;

stmt\_list

: stmt

| stmt\_list stmt

| /\* empty \*/

;

stmt

: assign\_stmt

| loop\_stmt

| control\_stmt

| break\_stmt

| print\_stmt

| switch\_stmt

| return\_stmt

| block\_stmt

| var\_decl

;

assign\_stmt

: ID ISON expr

;

loop\_stmt

: DO AR\_AG stmt\_list DE\_AG WHILE AR\_PAREN condition DE\_PAREN ERWT

| FOR AR\_PAREN for\_1 ERWT condition\_1 ERWT for\_2 DE\_PAREN AR\_AG stmt\_list DE\_AG

;

for\_1

: var\_decl

| assign\_stmt

| expr

| /\* empty \*/

;

for\_2

: assign\_stmt

| expr

| /\* empty \*/

;

control\_stmt

: IF AR\_PAREN condition DE\_PAREN AR\_AG stmt\_list DE\_AG %prec LOWER\_THAN\_ELSE

| IF AR\_PAREN condition DE\_PAREN AR\_AG stmt\_list DE\_AG ELSE AR\_AG stmt\_list DE\_AG

;

switch\_stmt

: SWITCH AR\_PAREN expr DE\_PAREN AR\_AG case\_clauses default\_clause\_opt DE\_AG

;

case\_clauses

: /\* empty \*/

| case\_clauses case\_clause

;

case\_clause

: CASE expr ANKAT stmt\_list

;

default\_clause\_opt

: /\* empty \*/

| DEFAULT ANKAT stmt\_list

;

print\_stmt

: PRINT AR\_PAREN STR\_LIT print\_arg\_list DE\_PAREN ERWT

;

print\_arg\_list

: /\* empty \*/

| KOM expr

| print\_arg\_list KOM expr

;

return\_stmt

: RETURN expr\_opt ERWT

;

break\_stmt

: BREAK ERWT

;

block\_stmt

: AR\_AG stmt\_list DE\_AG

;

expr\_opt

: /\* empty \*/

| expr

;

condition

: expr rel\_op expr

| expr

;

condition\_1

: /\* empty \*/

| condition

;

binary\_expr

: expr binary\_op expr

;

unary\_expr

: unary\_op expr

;

expr

: literal

| ID

| binary\_expr

| unary\_expr

| method\_call

| obj\_create

;

rel\_op

: MIK

| MEG

| SINEP

| INEK

| MIKIS

| MEGIS

;

method\_call

: ID TEL ID AR\_PAREN arg\_list DE\_PAREN

;

obj\_create

: NEW ID AR\_PAREN DE\_PAREN

;

arg\_list

: /\* empty \*/

| expr

| arg\_list KOM expr

;

literal

: NUM

| CHAR\_LIT

| STR\_LIT

| BOOL\_LIT

;

binary\_op

: SIN

| MEION

| EPI

| DIA

| AND

| OR

;

unary\_op

: MEION

| "!"

;

data\_type

: INT

| CHAR

| DBL

| BOOL

| STR

| VOID

| ID

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Σφάλμα: %s στη γραμμή %d, τόκεν: %s\n", s, yylineno, yytext);

error++;

}

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

if (argc > 1) {

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

if (!file) {

fprintf(stderr, "Δεν μπόρεσα να ανοίξω το %s\n", argv[1]);

return 1;

}

yyin = file;

}

int result = yyparse();

if (error > 0) {

fprintf(stderr, "Η ανάλυση απέτυχε με %d σφάλματα.\n", error);

} else {

fprintf(stdout, "Η ανάλυση ολοκληρώθηκε με επιτυχία χωρίς σφάλματα.\n");

}

return result;

}

Παρακάτω παραθέτουμε screenshot και κώδικες από τις δοκιμές που κάναμε.

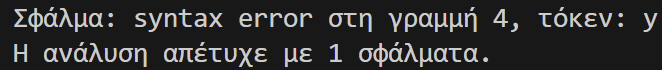
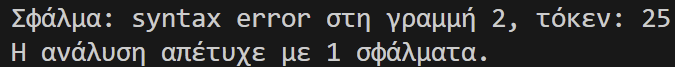
A screenshot of a computer program

Description automatically generatedΤο αρχείο που δοκιμάσαμε:



A screenshot of a computer program

Description automatically generatedΤο αρχείο που δοκιμάσαμε και έχει λάθος(δεν υπάρχει , στη γραμμή ):



**Ερώτημα 4**

Παρακάτω παραθέτουμε το τροποποιημένο bison στο οποίο δεν καταφέραμε να το υλοποιήσουμε πλήρως. Μας προέκυψαν κάποια προβλήματα με την εκτύπωση των λαθών οπότε σκεφτήκαμε να τα αποθηκεύσουμε σε ένα αρχείο txt. Ακόμα, δημιουργήσαμε τρείς νέες λειτουργείες για την τοποθέτηση του λάθους σε μια λίστα, για την καταγραφή του στο αρχείο και για να καθαρίσουμε τη λίστα. Τέλος, προσθέσαμε αυτή τη γραμμή στο statement για να κάνουμε recover το error.

Bison

%{

#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 Error {

int line;

char \*message;

struct Error \*next;

} Error;

Error \*error\_l = NULL;

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

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

n\_error->line = line;

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

n\_error->next = error\_l;

error\_l = n\_error;

}

void write\_errors\_to\_file() {

FILE \*file = fopen("errors.txt", "w");

if (file == NULL) {

fprintf(stderr, "Δεν μπόρεσα να ανοίξω το errors.txt.\n");

return;

}

Error \*current = error\_l;

while (current != NULL) {

fprintf(file, "Σφάλμα: %s στη γραμμή %d\n", current->message, current->line);

current = current->next;

}

fclose(file);

}

// Function to clear the error list

void clear\_errors() {

Error \*current = error\_l;

while (current != NULL) {

Error \*next = current->next;

free(current->message);

free(current);

current = next;

}

error\_l = NULL;

}

%}

%union {

char \*str;

double num;

}

%token <str> CHAR\_LIT

%token <str> STR\_LIT

%token <num> NUM

%token <str> UPPER\_ID

%token <str> ID

%token INT

%token CHAR

%token DBL

%token BOOL

%token STR

%token PUB

%token PRIV

%token CLS

%token NEW

%token RETURN

%token VOID

%token IF

%token ELSE

%token WHILE

%token DO

%token FOR

%token SWITCH

%token CASE

%token DEFAULT

%token BREAK

%token BOOL\_LIT

%token PRINT

%token AR\_PAREN

%token DE\_PAREN

%token AR\_AG

%token DE\_AG

%token ERWT

%token KOM

%token ISON

%token SIN

%token MEION

%token SINEP

%token INEK

%token MIK

%token MEG

%token MIKIS

%token MEGIS

%token AND

%token OR

%token EPI

%token DIA

%token TEL

%token ANKAT

%nonassoc LOWER\_THAN\_ELSE

%nonassoc ELSE

%left OR

%left AND

%left SINEP INEK

%left MIK MEG

%left SIN MEION

%left EPI DIA

%right ISON

%start prog

%%

prog

: class\_decl\_list

;

class\_decl\_list

: class\_decl

| class\_decl\_list class\_decl

;

class\_decl

: PUB CLS UPPER\_ID AR\_AG class\_body DE\_AG

| CLS UPPER\_ID AR\_AG class\_body DE\_AG

;

class\_body

: var\_decl\_list method\_decl\_list

;

var\_decl\_list

: var\_decl

| var\_decl\_list var\_decl

| /\* empty \*/

;

var\_decl

: data\_type var\_declrs ERWT

| access\_mod data\_type var\_declrs ERWT

;

var\_declrs

: var\_declr

| var\_declrs KOM var\_declr

;

var\_declr

: ID

| ID ISON expr

;

access\_mod

: PUB

| PRIV

;

method\_decl\_list

: method\_decl

| method\_decl\_list method\_decl

| /\* empty \*/

;

method\_decl

: access\_mod data\_type ID AR\_PAREN param\_list DE\_PAREN AR\_AG method DE\_AG

| data\_type ID AR\_PAREN param\_list DE\_PAREN AR\_AG method DE\_AG

;

param\_list

: param

| param\_list KOM param

| /\* empty \*/

;

param

: data\_type ID

;

method

: stmt\_list

;

stmt\_list

: stmt

| stmt\_list stmt

| /\* empty \*/

;

stmt

: assign\_stmt

| loop\_stmt

| control\_stmt

| break\_stmt

| print\_stmt

| switch\_stmt

| return\_stmt

| block\_stmt

| var\_decl

| error { add\_error(yylineno, "syntax error"); yyerrok; }

;

assign\_stmt

: ID ISON expr

;

loop\_stmt

: DO AR\_AG stmt\_list DE\_AG WHILE AR\_PAREN condition DE\_PAREN ERWT

| FOR AR\_PAREN for\_1 ERWT condition\_1 ERWT for\_2 DE\_PAREN AR\_AG stmt\_list DE\_AG

;

for\_1

: var\_decl

| assign\_stmt

| expr

| /\* empty \*/

;

for\_2

: assign\_stmt

| expr

| /\* empty \*/

;

control\_stmt

: IF AR\_PAREN condition DE\_PAREN AR\_AG stmt\_list DE\_AG %prec LOWER\_THAN\_ELSE

| IF AR\_PAREN condition DE\_PAREN AR\_AG stmt\_list DE\_AG ELSE AR\_AG stmt\_list DE\_AG

;

switch\_stmt

: SWITCH AR\_PAREN expr DE\_PAREN AR\_AG case\_clauses default\_clause\_opt DE\_AG

;

case\_clauses

: /\* empty \*/

| case\_clauses case\_clause

;

case\_clause

: CASE expr ANKAT stmt\_list

;

default\_clause\_opt

: /\* empty \*/

| DEFAULT ANKAT stmt\_list

;

print\_stmt

: PRINT AR\_PAREN STR\_LIT print\_arg\_list DE\_PAREN ERWT

;

print\_arg\_list

: /\* empty \*/

| KOM expr

| print\_arg\_list KOM expr

;

return\_stmt

: RETURN expr\_opt ERWT

;

break\_stmt

: BREAK ERWT

;

block\_stmt

: AR\_AG stmt\_list DE\_AG

;

expr\_opt

: /\* empty \*/

| expr

;

condition

: expr rel\_op expr

| expr

;

condition\_1

: /\* empty \*/

| condition

;

binary\_expr

: expr binary\_op expr

;

unary\_expr

: unary\_op expr

;

expr

: literal

| ID

| binary\_expr

| unary\_expr

| method\_call

| obj\_create

;

rel\_op

: MIK

| MEG

| SINEP

| INEK

| MIKIS

| MEGIS

;

method\_call

: ID TEL ID AR\_PAREN arg\_list DE\_PAREN

;

obj\_create

: NEW ID AR\_PAREN DE\_PAREN

;

arg\_list

: /\* empty \*/

| expr

| arg\_list KOM expr

;

literal

: NUM

| CHAR\_LIT

| STR\_LIT

| BOOL\_LIT

;

binary\_op

: SIN

| MEION

| EPI

| DIA

| AND

| OR

;

unary\_op

: MEION

| "!"

;

data\_type

: INT

| CHAR

| DBL

| BOOL

| STR

| VOID

| ID

;

%%

void yyerror(const char \*s) {

add\_error(yylineno, s); // Store the error instead of printing immediately

error\_count++;

}

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

if (argc > 1) {

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

if (!file) {

fprintf(stderr, "Δεν μπόρεσα να ανοίξω το %s\n", argv[1]);

return 1;

}

yyin = file;

}

int result = yyparse();

write\_errors\_to\_file(); // Write all errors to file at the end

if (error\_count > 0) {

fprintf(stderr, "Η ανάλυση απέτυχε με %d σφάλματα.\n", error\_count);

} else {

fprintf(stdout, "Η ανάλυση ολοκληρώθηκε με επιτυχία χωρίς σφάλματα.\n");

}

clear\_errors(); // Clear errors at the end

return result;

}

Παρακάτω παραθέτουμε screenshot και κώδικες από τις δοκιμές που κάναμε.

Το αρχείο που δοκιμάσαμε και έχει λάθος(δεν έχει γραφεί σωστά το public στη γραμμή 3 και δεν υπάρχει , στη γραμμή 4 ):

A screenshot of a computer program

Description automatically generated

Ο χρήστης αντικρίζει αυτό:

Καθώς και ένα νέο αρχείο errors.txt:

A black background with white text

Description automatically generated