Εξαμηνιαία Εργασία Μαθήματος :"Αρχές γλωσσών προγραμματισμού και Μεταφραστών"

Τομέας Λογικού των Υπολογιστών

Μέλη Ομάδας :

Βασιλείου Χαράλαμπος, 1043757, 7ο, cvasileiou@ceid.upatras.gr

Κολοκυθάς Ελευθέριος-Γεράσιμος, 1058118, 5ο, ekolokythas@ceid.upatras.gr

Τασιόπουλος Βασίλειος, 1057778, 5ο, tasi@ceid.upatras.gr

Τζόλας Χρήστος,1047072, 6ο, ctzolas@ceid.upatras.gr

Περιγραφή της γραμματικής της γλώσσας σε BNF

programm:

 program_start function main_part

| program_start main_part

| program_start struction function main_part

| program_start struction main_part

;

program_start : PROGRAM name

;

name : IDENTIFIERS

;

struction :

STRUCT name func-mainp-str_vars ENDSTRUCT

| TYPEDEF STRUCT name func-mainp-str_vars struction_end

;

function : FUNCTION name OPEN_PAR function_parameters CLOSE_PAR func-mainp-str_vars commands func_end

;

func-mainp-str_vars : VARS variables SEMICLN

;

commands : cmd

| commands cmd

;

cmd :

stmt

|PR_RETURN expr SEMICLN

|PR_CONT SEMICLN

|PR_BREAK SEMICLN

;

func_end : PR_RETURN end_func_value END_FUNCTION

```
;
main_part : PR_STARTMAIN OPEN_PAR CLOSE_PAR OPEN_HK func-mainp-str_vars
commands CLOSE_HK PR_ENDMAIN
| PR_STARTMAIN OPEN_PAR CLOSE_PAR OPEN_HK commands CLOSE_HK
PR_ENDMAIN
;
struction_end : name ENDSTRUCT
;
function_parameters:
type IDENTIFIERS
| type IDENTIFIERS COMMA function_parameters
;
variables : type list_var
;
type:
INTEGER
|CHAR
;
list_var :
name
|name Declare_Arrays
|name COMMA list_var
;
Declare_Arrays:
OPEN_BR list_var CLOSE_BR
;
end_func_value : name
| expr
;
stmt :
PR_PRINT OPEN_PAR DBL_QUOTE expr DBL_QUOTE OPEN_BR list_var CLOSE_BR
CLOSE_PAR SEMICLN
| PR_PRINT OPEN_PAR DBL_QUOTE expr DBL_QUOTE CLOSE_PAR SEMICLN
```

| PR_SWITCH OPEN_PAR expr CLOSE_PAR case_comms dflt_comms PR_ENDSWITCH

| PR_SWITCH OPEN_PAR expr CLOSE_PAR case_comms PR_ENDSWITCH

| PR_WHILE OPEN_PAR expr CLOSE_PAR commands PR_ENDWHILE

| PR_FOR name CLN EQUAL INTEGERS PR_TO INTEGERS PR_STEP INTEGERS commands expr PR_ENDFOR

| name EQUAL expr SEMICLN

| PR_PRINT OPEN_PAR DBL_QUOTE else_expr DBL_QUOTE OPEN_BR list_var CLOSE_BR CLOSE_PAR SEMICLN

| PR_PRINT OPEN_PAR DBL_QUOTE else_expr DBL_QUOTE CLOSE_PAR SEMICLN

| PR_SWITCH OPEN_PAR else_expr CLOSE_PAR case_comms dflt_comms PR_ENDSWITCH

| PR_SWITCH OPEN_PAR else_expr CLOSE_PAR case_comms PR_ENDSWITCH

| PR_WHILE OPEN_PAR else_expr CLOSE_PAR commands PR_ENDWHILE

| PR_FOR name CLN EQUAL INTEGERS PR_TO INTEGERS PR_STEP INTEGERS commands else_expr PR_ENDFOR

| PR_IF OPEN_PAR else_expr CLOSE_PAR PR_THEN commands elf_comms el_comms PR_ENDIF

| PR_IF OPEN_PAR else_expr CLOSE_PAR PR_THEN commands PR_ENDIF

| name EQUAL else_expr SEMICLN

;

case_comms :

PR_CASE OPEN_PAR expr CLOSE_PAR CLN commands

| PR_CASE OPEN_PAR expr CLOSE_PAR CLN commands case_comms

| PR_CASE OPEN_PAR else_expr CLOSE_PAR CLN commands

| PR_CASE OPEN_PAR else_expr CLOSE_PAR CLN commands case_comms

;

dflt_comms :

PR_DEFAULT CLN commands

;

elf_comms :

PR_ELSEIF commands

| PR_ELSEIF commands expr elf_comms

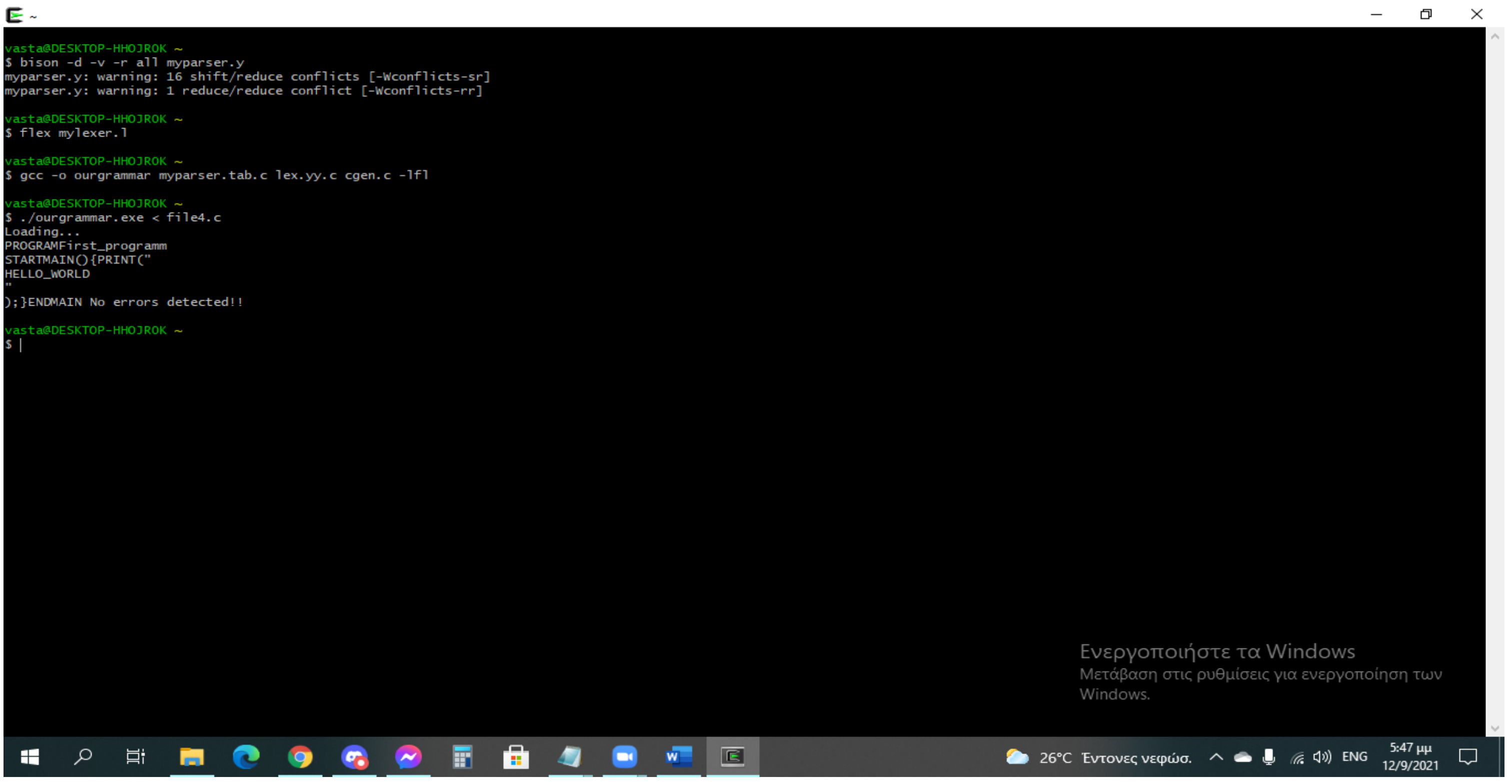
| PR_ELSEIF commands else_expr elf_comms

;

```
el_comms :
PR_ELSE commands
;
expr :
variables
|IDENTIFIERS
|IDENTIFIERS OPEN_PAR expr CLOSE_PAR
|OPEN_PAR expr CLOSE_PAR
|INTEGERS
|FLOATS
|CHARACTER
|expr PLUS expr
|expr SUB expr
|expr MUL expr
|expr DIV expr
;
else_expr :
expr DBL_EQUAL expr
|expr GREQUAL expr
|expr LSEQUAL expr
|expr GRTHAN expr
|expr LSTHAN expr
|expr INEQ expr
|expr AND expr
|expr S_AND expr
|expr OR expr
|expr S_OR expr
|expr NOT expr
|expr S_NOT expr
;
```

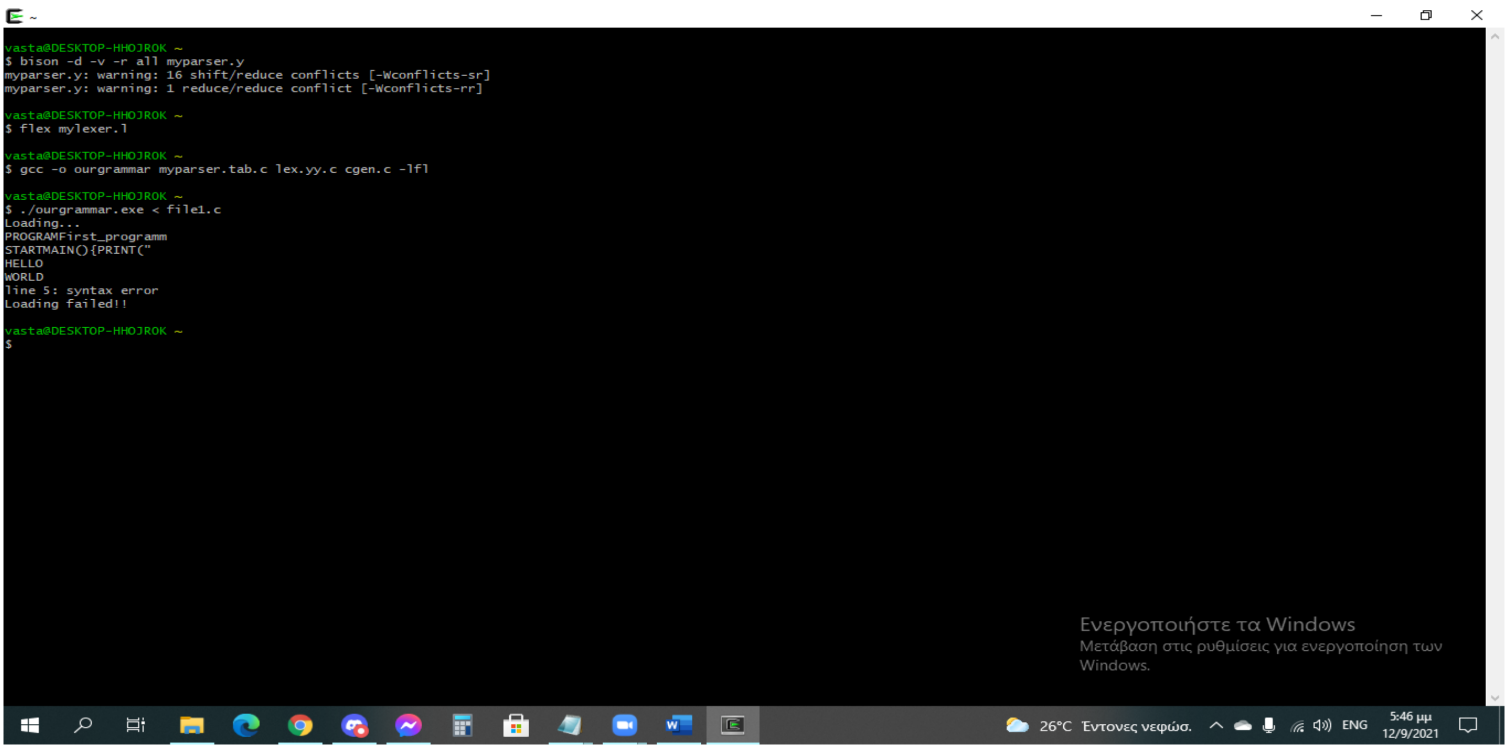
Screenshoots παραδειγμάτων εφαρμογής

- *Για ερώτημα 1*
  - *Σωστό παράδειγμα*



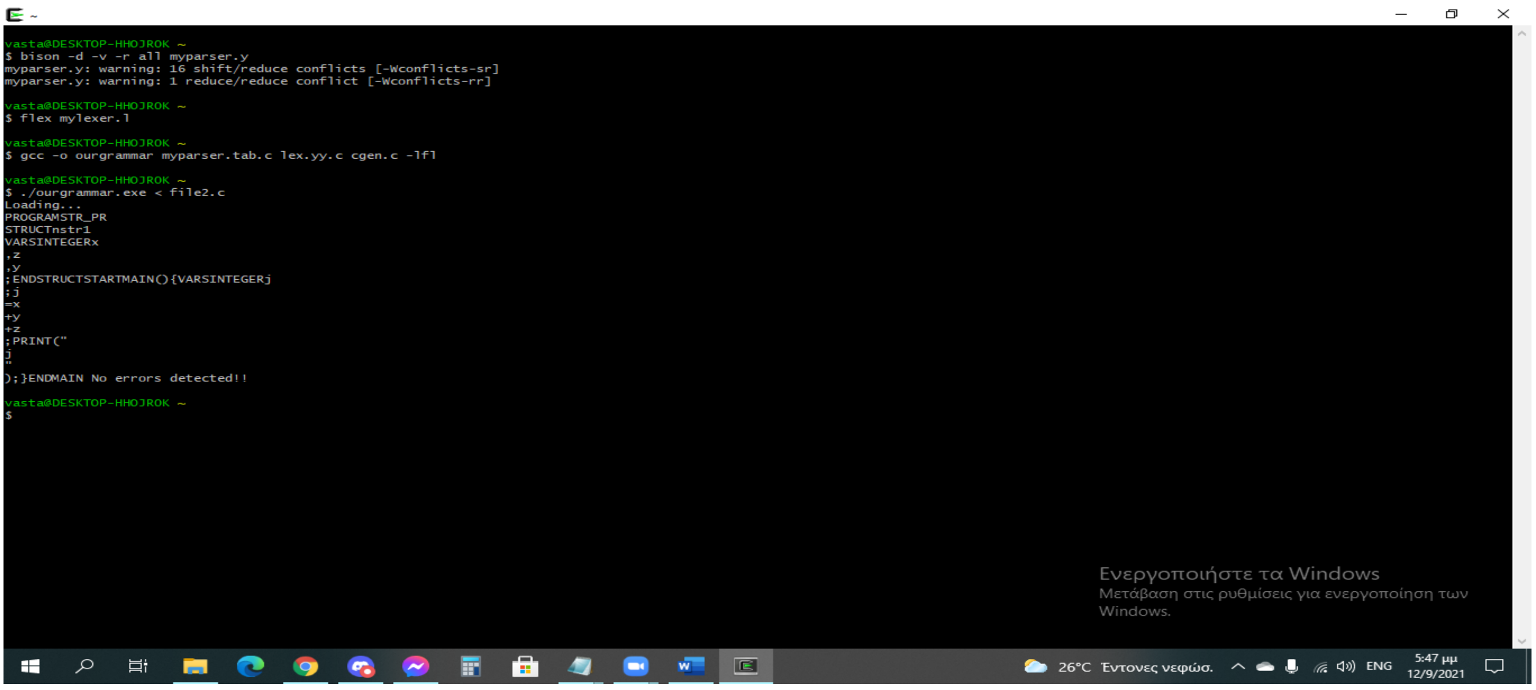
```
vasta@DESKTOP-HHOJROK ~
$ bison -d -v -r all myparser.y
myparser.y: warning: 16 shift/reduce conflicts [-Wconflicts-sr]
myparser.y: warning: 1 reduce/reduce conflict [-Wconflicts-rr]

vasta@DESKTOP-HHOJROK ~
$ flex mylexer.l

vasta@DESKTOP-HHOJROK ~
$ gcc -o ourgrammar myparser.tab.c lex.yy.c cgen.c -lfl

vasta@DESKTOP-HHOJROK ~
$ ./ourgrammar.exe < file4.c
Loading...
PROGRAMFirst_programm
STARTMAIN(){PRINT("
HELLO_WORLD
"
);}ENDMAIN No errors detected!!

vasta@DESKTOP-HHOJROK ~
$ |
```

  - *Λάθος παράδειγμα*



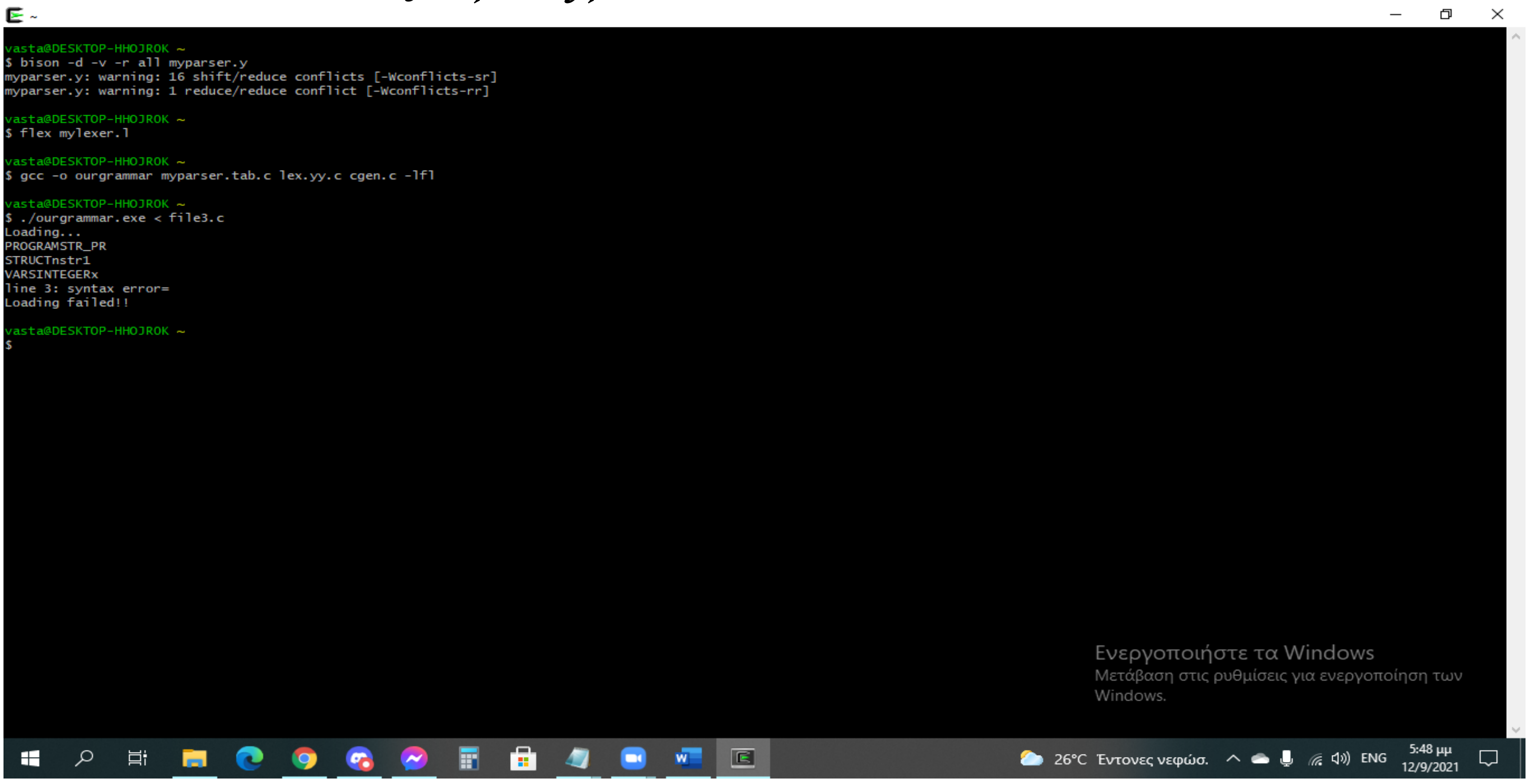
```
vasta@DESKTOP-HHOJROK ~
$ bison -d -v -r all myparser.y
myparser.y: warning: 16 shift/reduce conflicts [-Wconflicts-sr]
myparser.y: warning: 1 reduce/reduce conflict [-Wconflicts-rr]

vasta@DESKTOP-HHOJROK ~
$ flex mylexer.l

vasta@DESKTOP-HHOJROK ~
$ gcc -o ourgrammar myparser.tab.c lex.yy.c cgen.c -lfl

vasta@DESKTOP-HHOJROK ~
$ ./ourgrammar.exe < file1.c
Loading...
PROGRAMFirst_programm
STARTMAIN(){PRINT("
HELLO
WORLD
line 5: syntax error
Loading failed!!

vasta@DESKTOP-HHOJROK ~
$
```

- *Για ερώτημα 2*
  - *Σωστό παράδειγμα*



```
vasta@DESKTOP-HHOJROK ~
$ bison -d -v -r all myparser.y
myparser.y: warning: 16 shift/reduce conflicts [-Wconflicts-sr]
myparser.y: warning: 1 reduce/reduce conflict [-Wconflicts-rr]

vasta@DESKTOP-HHOJROK ~
$ flex mylexer.l

vasta@DESKTOP-HHOJROK ~
$ gcc -o ourgrammar myparser.tab.c lex.yy.c cgen.c -lfl

vasta@DESKTOP-HHOJROK ~
$ ./ourgrammar.exe < file2.c
Loading...
PROGRAMSTR_PR
STRUCTnstr1
VARSINTEGERx
,z
,y
;ENDSTRUCTSTARTMAIN(){VARSINTEGERj
;j
=x
+y
+z
;PRINT("
j
"
);}ENDMAIN No errors detected!!

vasta@DESKTOP-HHOJROK ~
$
```

  - *Λάθος παράδειγμα*

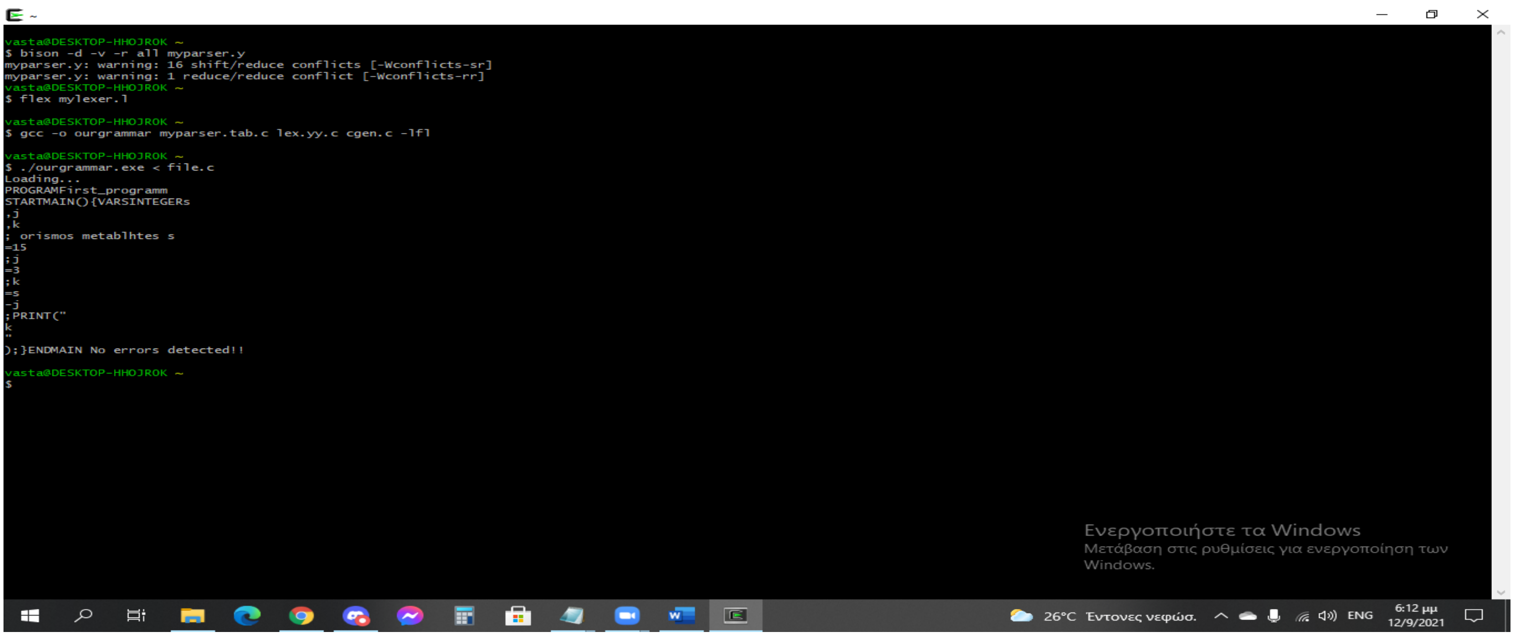


```
vasta@DESKTOP-HHOJROK ~
$ bison -d -v -r all myparser.y
myparser.y: warning: 16 shift/reduce conflicts [-Wconflicts-sr]
myparser.y: warning: 1 reduce/reduce conflict [-Wconflicts-rr]

vasta@DESKTOP-HHOJROK ~
$ flex mylexer.l

vasta@DESKTOP-HHOJROK ~
$ gcc -o ourgrammar myparser.tab.c lex.yy.c cgen.c -lfl

vasta@DESKTOP-HHOJROK ~
$ ./ourgrammar.exe < file3.c
Loading...
PROGRAMSTR_PR
STRUCTnstr1
VARSINTEGERx
line 3: syntax error=
Loading failed!!

vasta@DESKTOP-HHOJROK ~
$
```

❖ *Για ερώτημα 4*

Από το παρακάτω πρόγραμμα προκύπτει :

```
PROGRAM First_programm
STARTMAIN()
{
VARS INTEGER s, j, k;  /* orismos metablhtes */

    s = 15;
    j = 3;
    k = s - j;

    PRINT("k");
}

ENDMAIN
```



```
vasta@DESKTOP-HHOJROK ~
$ bison -d -v -r all myparser.y
myparser.y: warning: 16 shift/reduce conflicts [-Wconflicts-sr]
myparser.y: warning: 1 reduce/reduce conflict [-Wconflicts-rr]
vasta@DESKTOP-HHOJROK ~
$ flex mylexer.l

vasta@DESKTOP-HHOJROK ~
$ gcc -o ourgrammar myparser.tab.c lex.yy.c cgen.c -lfl

vasta@DESKTOP-HHOJROK ~
$ ./ourgrammar.exe < file.c
Loading...
PROGRAMFirst_programm
STARTMAIN(){VARSINTEGERs
,j
,k
; orismos metablhtes s
=15
;j
=3
;k
=s
-j
;PRINT("
k
"
);}ENDMAIN No errors detected!!

vasta@DESKTOP-HHOJROK ~
$
```

Ερώτημα 3.

Στα παραπάνω screenshots εμφανίζονται και τα ανάλογα μηνύματα για σωστή ή λανθασμένη χρήση του συντακτικού.

```
 int main(int argc, char *argv[]){
   ++argv; --argc;
   int parser_return_value = 0;
   if (argc==1) {
      FILE *file_pointer = fopen(argv[0],"r");
      if (file_pointer!=NULL) {
         yyin = file_pointer;
         parser_return_value = yyparse();
      }
      else {
         printf("Error!!!\n");
         return 1;
      }
   }
   else {
        printf ("Loading...\n");
        parser_return_value = yyparse();
   }
   if (parser_return_value==0) {
      printf(" No errors detected!!\n");
   }
   else {
      printf("\nLoading failed!!\n");
   }
   return 0;
}
```

Ερώτημα 4.

Στο αρχείο mylexer.l στις σειρες 31-35 έχουμε ορίσει τις μορφές των σχολίων που θα δέχεται η γραμματική μας.

```
 "/*"          { BEGIN(C_COMMENT); }
<C_COMMENT>"*/"    { BEGIN(INITIAL); }
<C_COMMENT>"/*"[\n.]*"*/"
```

Παράρτημα

1. myparser.y

```
%{
  #include <stdio.h>
  #include <stdlib.h>
  #include <unistd.h>
  #include <ctype.h>
  #include "cgen.h"

  int yylex();
  extern int lineNum;
  extern FILE *yyin;
  extern FILE *yyout;
  extern int yylineno;
%}

%union
{
	char* str;
  int num;
}

%define parse.trace
%debug
```

```
%token PROGRAM FUNCTION VARS PR_INTEGER PR_CHAR
END_FUNCTION PR_RETURN PR_STARTMAIN PR_ENDMAIN
STRUCT ENDSTRUCT TYPEDEF

%token PR_IF PR_THEN PR_ENDIF PR_ELSEIF PR_ELSE PR_FOR
PR_STEP PR_TO PR_ENDFOR PR_WHILE PR_ENDWHILE
PR_SWITCH PR_CASE PR_DEFAULT PR_ENDSWITCH

%token PR_END PR_BEGIN PR_PRINT PR_BREAK PR_CONT
INTEGER CHAR

%token NW_LINE EQUAL CLN DBL_QUOTE SEMICLN COMMA
OPEN_BR CLOSE_BR OPEN_HK CLOSE_HK OPEN_PAR
CLOSE_PAR

%token PLUS SUB MUL DIV DBL_EQUAL GREQUAL LSEQUAL
GRTHAN LSTHAN INEQ AND S_AND OR S_OR NOT S_NOT


%token <str> IDENTIFIERS

%token <str> INTEGERS

%token <str> FLOATS


%type <str> expr

%type <str> type

%type <str> list_var

%type <str> Declare_Arrays

%type <str> function_parameters

%type <str> cmd

%type <str> programm

%type <str> stmt

%type <str> program_start

%type <str> name

%type <str> main_part

%type <str> function
```

```
%type <str> func-mainp-str_vars
%type <str> commands
%type <str> func_end
%type <str> variables
%type <str> end_func_value
%type <str> case_comms
%type <str> dflt_comms
%type <str> elf_comms
%type <str> el_comms
%type <str> struction
%type <str> struction_end
%type <str> else_expr
%%

programm:
 program_start function main_part              { $$ = template("%s\n %s\n
%s\n", $1,$2,$3);}
| program_start main_part                        { $$ = template("%s\n
%s\n",$1,$2);}
| program_start struction function main_part     { $$ = template("%s\n
%s\n %s\n %s\n", $1,$2,$3,$4);}
| program_start struction main_part             { $$ = template("%s\n %s\n
%s\n",$1,$2,$3);}
;

program_start : PROGRAM name                     { $$ =
template("PROGRAM %s\n", $2);}
;
```

```
name : IDENTIFIERS { $$ = template("%s\n");}

;


struction :

STRUCT name func-mainp-str_vars ENDSTRUCT { $$ =
template("STRUCT %s\n %s\n ENDSTRUCT", $2,$3);}

| TYPEDEF STRUCT name func-mainp-str_vars struction_end { $$ =
template("TYPEDEF STRUCT %s\n %s\n %s\n", $3,$4,$5);}

;


function : FUNCTION name OPEN_PAR function_parameters
CLOSE_PAR func-mainp-str_vars commands func_end { $$ =
template("FUNCTION %s (%s) \n %s\n %s\n %s\n", $2,$4,$6,$7,$8);}

;


func-mainp-str_vars : VARS variables SEMICLN { $$ =
template("VARS %s;\n", $2);}

;


commands : cmd { $$ = template("%s\n", $1);}

| commands cmd { $$ = template("%s\n %s\n", $1,$2);}

;


cmd :

stmt                                    { $$ = template("%s\n",$1); }

|PR_RETURN expr SEMICLN                 { $$ = template("RETURN
%s\n;",$2); }

|PR_CONT SEMICLN                        { $$ = template("CONTINUE;\n");
}

|PR_BREAK SEMICLN                       { $$ = template("BREAK;\n"); }
```

```
;

func_end : PR_RETURN end_func_value END_FUNCTION { $$ =
template("RETURN %s\n END_FUNCTION",$2); }

;

main_part : PR_STARTMAIN OPEN_PAR CLOSE_PAR OPEN_HK
func-mainp-str_vars commands CLOSE_HK PR_ENDMAIN { $$ =
template("STARTMAIN() \n {\n %s\n %s\n }\n ENDMAIN", $5,$6);}

| PR_STARTMAIN OPEN_PAR CLOSE_PAR OPEN_HK commands
CLOSE_HK PR_ENDMAIN { $$ = template("STARTMAIN() \n {\n
%s\n }\n ENDMAIN", $5);}

;

struction_end : name ENDSTRUCT { $$ = template("%s
ENDSTRUCT", $1);}

;

function_parameters:

type IDENTIFIERS { $$ = template("%s %s", $1);}

| type IDENTIFIERS COMMA function_parameters   { $$ =
template("%s %s, %s", $1,$4);}

;

variables : type list_var { $$ = template("%s %s", $1,$2);}

;

type:

INTEGER                    { $$ = template("INTEGER"); }

|CHAR                      { $$ = template("CHAR"); }
```

```
;


;
list_var :
name  { $$ = template("%s", $1);}
|name Declare_Arrays { $$ = template("%s %s", $1,$2);}
|name COMMA list_var { $$ = template("%s, %s", $1,$3);}
;

Declare_Arrays:
OPEN_BR list_var CLOSE_BR { $$ = template("[%s]\n", $2);}
;

end_func_value : name { $$ = template("%s");}
| expr { $$ = template("%s");}
;
stmt :
PR_PRINT OPEN_PAR DBL_QUOTE expr DBL_QUOTE OPEN_BR
list_var CLOSE_BR CLOSE_PAR SEMICLN { $$ =
template("PRINT(%s%s%s[%s]);\n", $4, $7); }
| PR_PRINT OPEN_PAR DBL_QUOTE expr DBL_QUOTE
CLOSE_PAR SEMICLN { $$ = template("PRINT(%s%s%s);\n", $4); }
| PR_SWITCH OPEN_PAR expr CLOSE_PAR case_comms dflt_comms
PR_ENDSWITCH { $$ = template("SWITCH(%s)\n %s\n %s\n
ENDSWITCH\n", $3, $5, $6); }
| PR_SWITCH OPEN_PAR expr CLOSE_PAR case_comms
PR_ENDSWITCH { $$ = template("SWITCH(%s)\n %s\n
ENDSWITCH\n", $3, $5); }
```

```
| PR_WHILE OPEN_PAR expr CLOSE_PAR commands
PR_ENDWHILE { $$ = template("WHILE (%s)\n %s\n
ENDWHILE\n",$3,$5); }

| PR_FOR name CLN EQUAL INTEGERS PR_TO INTEGERS
PR_STEP INTEGERS commands expr PR_ENDFOR {$$ =
template("FOR %s:=%s TO %s STEP %s\n %s\n %s\n ENDFOR\n",
$2,$10,$11);}

| name EQUAL expr SEMICLN  { $$ = template("%s=%s;\n",$1,$3);}

| PR_PRINT OPEN_PAR DBL_QUOTE else_expr DBL_QUOTE
OPEN_BR list_var CLOSE_BR CLOSE_PAR SEMICLN { $$ =
template("PRINT(%s%s%s[%s]);\n", $4, $7); }

| PR_PRINT OPEN_PAR DBL_QUOTE else_expr DBL_QUOTE
CLOSE_PAR SEMICLN { $$ = template("PRINT(%s%s%s);\n", $4); }

| PR_SWITCH OPEN_PAR else_expr CLOSE_PAR case_comms
dflt_comms PR_ENDSWITCH { $$ = template("SWITCH(%s)\n %s\n
%s\n ENDSWITCH\n", $3, $5, $6); }

| PR_SWITCH OPEN_PAR else_expr CLOSE_PAR case_comms
PR_ENDSWITCH { $$ = template("SWITCH(%s)\n %s\n
ENDSWITCH\n", $3, $5); }

| PR_WHILE OPEN_PAR else_expr CLOSE_PAR commands
PR_ENDWHILE { $$ = template("WHILE (%s)\n %s\n
ENDWHILE'\n",$3,$5); }

| PR_FOR name CLN EQUAL INTEGERS PR_TO INTEGERS
PR_STEP INTEGERS commands else_expr PR_ENDFOR {$$ =
template("FOR %s:=%s TO %s STEP %s\n %s\n %s\n ENDFOR\n", $2,
$10,$11);}

| PR_IF OPEN_PAR else_expr CLOSE_PAR PR_THEN commands
elf_comms el_comms PR_ENDIF {$$ = template("IF (%s) THEN\n
%s\n %s\n %s\n ENDIF\n", $3,$6,$7,$8);}

| PR_IF OPEN_PAR else_expr CLOSE_PAR PR_THEN commands
PR_ENDIF {$$ = template("IF (%s) THEN\n %s\n ENDIF\n", $3, $6);}

| name EQUAL else_expr SEMICLN  { $$ = template("%s=%s;",$1,$3);
}

;
```

```
case_comms :
PR_CASE OPEN_PAR expr CLOSE_PAR CLN commands { $$ =
template("CASE(%s):\n %s",$3,$6); }
| PR_CASE OPEN_PAR expr CLOSE_PAR CLN commands
case_comms{ $$ = template("CASE(%s):\n %s\n %s",$3,$6,$7); }
| PR_CASE OPEN_PAR else_expr CLOSE_PAR CLN commands { $$ =
template("CASE(%s):\n %s",$3,$6); }
| PR_CASE OPEN_PAR else_expr CLOSE_PAR CLN commands
case_comms { $$ = template("CASE(%s):\n %s\n %s",$3,$6,$7); }
;

dflt_comms :
PR_DEFAULT CLN commands { $$ = template("DEFAULT:\n
%s",$3); }
;

elf_comms :
PR_ELSEIF commands { $$ = template("ELSEIF\n %s",$2); }
| PR_ELSEIF commands expr elf_comms { $$ = template("ELSEIF\n
%s\n %s\n %s",$2,$3,$4); }
| PR_ELSEIF commands else_expr elf_comms { $$ =
template("ELSEIF\n %s\n %s\n %s",$2,$3,$4); }
;

el_comms :
PR_ELSE commands { $$ = template("ELSE\n %s",$2); }
;
expr :
variables
```

```
|IDENTIFIERS                    { $$ = template("%s"); }
|IDENTIFIERS OPEN_PAR expr CLOSE_PAR    { $$ = template("%s
(%s)",$3); }
|OPEN_PAR expr CLOSE_PAR               { $$ = template("(%s)",$2);
}
|INTEGERS                          { $$ = template("%s"); }
|FLOATS                            { $$ = template("%s"); }
|expr PLUS expr                { $$ = template("%s + %s",$1,$3); }
|expr SUB expr                     { $$ = template("%s - %s",$1,$3);
}
|expr MUL expr                     { $$ = template("%s * %s",$1,$3);
}
|expr DIV expr                     { $$ = template("%s / %s",$1,$3);
}
;


else_expr :
expr DBL_EQUAL expr                { $$ = template("%s ==
%s",$1,$3); }
|expr GREQUAL expr                 { $$ = template("%s >=
%s",$1,$3); }
|expr LSEQUAL expr                 { $$ = template("%s <=
%s",$1,$3); }
|expr GRTHAN expr                { $$ = template("%s > %s",$1,$3); }
|expr LSTHAN expr                  { $$ = template("%s < %s",$1,$3);
}
|expr INEQ expr                { $$ = template("%s != %s",$1,$3); }
|expr AND expr                   { $$ = template("%s and
%s",$1,$3); }
|expr S_AND expr                   { $$ = template("%s &&
%s",$1,$3); }
```

```
|expr OR expr                                { $$ = template("%s or
%s",$1,$3); }
|expr S_OR expr                         { $$ = template("%s || %s",$1,$3); }
|expr NOT expr                               { $$ = template("%s not
%s",$1,$3); }
|expr S_NOT expr                        { $$ = template("%s ! %s",$1,$3); }
;


%%


int main(int argc, char *argv[]){
    ++argv; --argc;
    int parser_return_value = 0;
    if (argc==1) {
        FILE *file_pointer = fopen(argv[0],"r");
        if (file_pointer!=NULL) {
            yyin = file_pointer;
            parser_return_value = yyparse();
        }
        else {
            printf("Error!!!\n");
            return 1;
        }
    }
    else {
          printf ("Loading...\n");
          parser_return_value = yyparse();
    }
    if (parser_return_value==0) {
        printf(" No errors detected!!\n");
    }
    else {
        printf("\nLoading failed!!\n");
    }
    return 0;
}
```

2. mylexer.l

```
%{
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  #include <errno.h>
  #include "cgen.h"
  #include  "myparser.tab.h"

  extern int yylex();
  int lineNum = 1;

%}

%option yylineno



/* definitions */



%x C_COMMENT

Identifiers [A-Za-z][A-Za-z0-9_]*
Integers [0-9][1-9]*
Floats {Integers}+("."{Integers}+)?([eE][+-]?{Integers})?
DBL_QUOTE ["]?
```

```
/* rules */

%%

"/*"            { BEGIN(C_COMMENT); }
<C_COMMENT>"*/" { BEGIN(INITIAL); }
<C_COMMENT>"/*"[\n.]*"*/"

"WHILE"         { printf("WHILE"); return PR_WHILE;}
"IF"            { printf("IF"); return PR_IF;}
"RETURN"        { printf("RETURN"); return PR_RETURN;}
"BREAK"             { printf("BREAK"); return PR_BREAK;}
"CONTINUE"      { printf("CONTINUE"); return PR_CONT;}
"FOR"               { printf("FOR"); return PR_FOR;}
"CHARACTER"     { printf("CHARACTER"); return PR_CHAR;}
"END"           { printf("END"); return PR_END;}
"BEGIN"             { printf("BEGIN"); return PR_BEGIN;}
"PROGRAM"       { printf("PROGRAM"); return PROGRAM;}
"FUNCTION"      { printf("FUNCTION"); return FUNCTION;}
"VARS"              { printf("VARS"); return VARS;}
"STARTMAIN"     { printf("STARTMAIN"); return PR_STARTMAIN;}
"ENDMAIN"       { printf("ENDMAIN"); return PR_ENDMAIN;}
"ENDWHILE"      { printf("ENDWHILE"); return PR_ENDWHILE;}
"ENDFOR"            { printf("ENDFOR"); return PR_ENDFOR;}
"ENDIF"     { printf("ENDIF"); return PR_ENDIF;}
"ELSEIF"    { printf("ELSEIF"); return PR_ELSEIF;}
```

```
"ELSE"              { printf("ELSE"); return PR_ELSE;}
"THEN"              { printf("THEN"); return PR_THEN;}
"CASE"      { printf("CASE"); return PR_CASE;}
"DEFAULT"           { printf("DEFAULT"); return PR_DEFAULT;}
"SWITCH"            { printf("SWITCH"); return PR_SWITCH;}
"ENDSWITCH"   { printf("ENDSWITCH"); return PR_ENDSWITCH;}
"END_FUNCTION"      { printf("END_FUNCTION"); return
END_FUNCTION;}
"PRINT"     { printf("PRINT"); return PR_PRINT;}
"TO"        { printf("TO"); return PR_TO;}
"STEP"              { printf("STEP"); return PR_STEP;}
"STRUCT"            { printf("STRUCT"); return STRUCT;}
"ENDSTRUCT"   { printf("ENDSTRUCT"); return ENDSTRUCT;}
"TYPEDEF"           { printf("TYPEDEF"); return TYPEDEF;}
"INTEGER"     { printf("INTEGER"); return INTEGER;}
"CHAR"      { printf("CHAR"); return CHAR;}



"+"         { printf("+"); return PLUS;}
"-"         { printf("-"); return SUB;}
"*"         { printf("*"); return MUL;}
"/"         { printf("/"); return DIV;}

"=="       { printf("=="); return DBL_EQUAL;}
">="       { printf(">="); return GREQUAL;}
"<="        { printf("<="); return LSEQUAL;}
">"         { printf(">"); return GRTHAN;}
"<"         { printf("<"); return LSTHAN;}
```

```
"!="          { printf("!="); return INEQ;}

"and"         { printf("and"); return AND;}
"&&"          { printf("&&"); return S_AND;}
"or"          { printf("or"); return OR;}
"||"          { printf("||"); return S_OR;}
"not"         { printf("not"); return NOT;}
"!"           { printf("!"); return S_NOT;}

";"           { printf(";"); return SEMICLN;}
"("           { printf("("); return OPEN_PAR;}
")"           { printf(")"); return CLOSE_PAR;}
","           { printf(","); return COMMA;}
"["           { printf("["); return OPEN_BR;}
"]"           { printf("]"); return CLOSE_BR;}
":"          { printf(":"); return CLN;}
"="          { printf("="); return EQUAL;}
"{"           { printf("{"); return OPEN_HK;}
"}"           { printf("}"); return CLOSE_HK;}
[\t]        {}
\n          lineNum++;

{Identifiers} { printf("%s \n", yytext); return IDENTIFIERS;}
{Integers} { printf("%s \n", yytext); return INTEGERS;}
{Floats} { printf("%s \n", yytext); return FLOATS;}
["]? { printf("%s \n", yytext); return DBL_QUOTE;}
```

```
[ \r\t]+                    /* skip whitespace */
.        { printf("Line %d Lexical Error: Unrecognized literal %s \n",
lineNum, yytext); }


%%


int yywrap(){
    return 1;
}
```

Σχόλια

1. *Για τη σωστή λειτουργία της γραμματικής δημιουργήσαμε και τα αρχεία "cgen.c" & "cgen.h".*
2. *Η γραμματική δεν αναγνωρίζει τα κενά διαστήματα και δεν εφαρμόζει σωστά την αλλαγή γραμμών.*