Θεοδωρακοπούλου Γεωργία-Μαρία, ΑΜ:1067492,Έτος:Γ', up1067492@upnet.gr
Κουϊμτζίδης Σάββας, ΑΜ:1059704,Έτος:Δ', up1059704@upnet.gr
Σκύρλα Αγάθη, ΑΜ:1064888, Έτος:Δ', up1064888@upnet.gr
Τυροβολά Αθανασία, ΑΜ:1064887, Έτος:Δ', up1064887@upnet.gr

Project
Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

1. Περιγραφή της Γραμματικής της γλώσσας σε BNF

```
<program>::= PROGRAM <id> <newline> <varexp> <struct> <function> <main> EOF

<character>::= [a-zA-Z]

<tab>::= [\t]

<newline>::= [\n]

<id>::= [a-z][a-z0-9]*

<digit> ::= [0-9]

<number> ::= [0-9][0-9]*

<data_type>::= <var>
              |<char>
              |<int>

<program-initialization> ::= PROGRAM

<struct>::= <struct_type> struct <newline>| ε

<struct_type>::= STRUCT <id> <newline> <varexp> <id> ENDSTRUCT
                |TYPEDEF STRUCT <id> <newline> <varexp> <id> ENDSTRUCT
                | ε

<function>::= FUNCT <ID> (<var>) <newline> <varexp> <functionbody> RETURN
ENDFUNCT <newline>
              |<function> <function>
              | ε

<varexp>::= VARS <var_type> <var>; <newline>
           |<varexp> <varexp>
           |ε

<vartype>::= CHAR
             |INTEGER

<var>::= <id>
        |<id>, <var>
        |ε

<functionbody>::= <command>
             |<functionbody> <command>
             | ε
```

```
<command>::= <assignment> <newline>
             |<loop> <newline>
             |<check> <newline>
             |<print> <newline>
             |<break> <newline>

<assignment>::= <id> = <expression>
             |<id> = <expression>;

<expression>::= <literal>
                |<id> (<var>)
                |<id> [T_NUMBER]
                |<operation>

<operation>::= <literal> <noperator> <literal>
               |<operation> <noperator> <operation>
               |<operation> <noperator> <literal>
               |<literal> <noperator> <operation>
               |(<operation>)

<literal>::= <number>
           |<id>

<noperator>::= +
              |-
              |^
              |*
              |/
              |=
              |%

<loop>::= <forloop>
        |<whileloop>

<forloop>::= FOR <id> = <number> TO <number> STEP <number><newline><function>
ENDFOR

<whileloop>::= WHILE (<condition>)  <newline> <function> ENDWHILE

<condition>::= <literal> <operators> <literal>

<operators>::= <loperator>
               |<coperator>

<loperator>::= <
              |>
              |==
```

```
              |!=


<coperator>::= AND
              |OR

<check>::= <checkif>
          |<checkcase>

<checkif>::= IF ( <condition> ) THEN <newline> <functionbody> ENDIF
           |IF (<condition>) THEN <newline> <functionbody> ELSE <newline>
<functionbody> ENDIF
           |IF (<condition>) THEN <newline> <functionbody> <elseif> ELSE
<newline>
            <functionbody> ENDIF

<elseif>::= ELSEIF <newline> <functionbody>
           |<elseif> <elseif>

<checkcase>::= SWITCH ( <expression> ) <newline> <case> <default> ESWITCH


<case>::= CASE ( <expression> ) : <newline> <functionbody>
         | <case> <case>


<default>::= DEFAULT : <newline> <functionbody>
            |ε


<print>::= PRINT (" <message> " , [<var>]) ;
           PRINT (" <message> ");


<message>::= < literal >
           |< literal > <message>
           |ε


<break>::= BREAK ;


<return>::= RETURN <literal> ; <newline>

<main>::= SMAIN <newline> <varexp> <functionbody> EMAIN
```

2. Τελικό FLEX αρχείο

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <math.h>
    #include "bison.tab.h"

    int lineno = 1;  /* that means that we start to count lines from 1*/
%}

 /* reads only one file*/
%option noyywrap
 /* shows the current inout line*/
%option yylineno
%x C_COMMENT




CHARACTER  [a-zA-Z]
DIGIT       [0-9]
NUMBER      [0-9][0-9]*
ID        [a-z][a-z0-9]*
NEWLINE          \n
TAB              \t



    /* -----TRANSLATION RULES  ---- */
%%

"PROGRAM"              {printf("Start the program \n"); return T_PROG;}
"STARTMAIN"            {printf("Start of main \n"); return T_SMAIN;}
"CHAR"                 {printf("Character \n");  return T_CHAR;}
"INTEGER"              {printf("Integer \n");return T_INTEGER;}
"VARS"                 {printf("Vars \n"); return T_VARS;}
"PRINT"                {printf("Print \n"); return T_PRINT;}
"STRUCT"               {printf("This is a struct \n"); return T_STRUCT;}
"ENDSTRUCT"            {printf("End of struct \n"); return T_ENDSTRUCT;}
"FUNCTION"             {printf("Start of function \n"); return T_FUNCT;}
"END_FUNCTION"         {printf("The end of function \n"); return T_ENDFUNC;}
"TYPEDEF"              {printf("Typedef \n"); return T_TYPEDEF;}
"BREAK"                {printf("Break \n"); return T_BREAK;}
"RETURN"               {printf("Returning... \n"); return T_RETURN;}
"IF"                   {printf("If \n"); return T_IF;}
```

```
"THEN"                  {printf("Then \n"); return T_THEN;}
"ELSE"                  {printf("Else \n"); return T_ELSE;}
"ELSEIF"                {printf("Else If \n"); return T_ELSEIF;}
"ENDIF"                 {printf("End of if \n"); return T_ENDIF;}
"WHILE"                 {printf("while \n"); return T_WHILE;}
"ENDWHILE"              {printf("End of while \n"); return T_ENDWHILE;}
"FOR"                   {printf("For \n"); return T_FOR;}
"TO"                    {printf("To \n"); return T_TO;}
"STEP"                  {printf("Step \n"); return T_STEP;}
"ENDFOR"                {printf("End of for \n"); return T_ENDFOR;}
"CASE"                  {printf("Case \n"); return T_CASE;}
"SWITCH"                {printf("Switch \n"); return T_SWITCH;}
"DEFAULT"               {printf("Default \n"); return T_DEF;}
"ENDSWITCH"             {printf("End of switch \n"); return T_ESWITCH;}
"ENDMAIN"               {printf("End of main \n"); return T_EMAIN;}

{NUMBER}                {printf("Number \n"); return T_NUMBER;}
{NEWLINE}               {printf("New Line \n"); return T_NEWLINE;}
{ID}                    {printf("ID \n"); return T_ID;}
{TAB}                   {printf("TAB"); return T_TAB;}

"||"                    {printf("Or \n"); return T_OROP;}
"&&"                    {printf("And \n"); return T_ANDOP;}
"=="                    {printf("Equal \n"); return T_EQUAL;}
"+"                     {printf("Plus \n"); return T_PLUS;}
"-"                     {printf("Minus \n"); return T_MINUS;}
"*"                     {printf("Multiply \n"); return T_MULTIPLY;}
"!"                     {printf("Exclamation \n"); return T_NOT;}
"("                     {printf("Left Parenth \n"); return T_LEPARENTH;}
")"                     {printf("Right Parenth \n"); return T_RIPARENTH;}
";"                     {printf("Semicolon \n"); return T_SEMICOL;}
"."                     {printf("Dot"); return T_DOT;}
","                     {printf("Comma \n"); return T_COMMA;}
"="                     {printf("Assign \n"); return T_ASSIGN;}
"!="                    {printf("Not equal \n"); return T_NOTEQUAL;}
"/"                     {printf("Divide \n"); return T_DIVIDE;}
"^"                     {printf("Power \n"); return T_POWER;}
"::"                    {printf("Meth \n"); return T_METH;}
":"                     {printf("Column \n"); return T_COL;}
"["                     {printf("Left Bracket \n"); return T_LEBRACKET;}
"]"                     {printf("Right Bracket \n"); return T_RIBRACKET;}
"&"                     {printf("Refer \n"); return T_REFER;}
"{"                     {printf("Left Brace \n"); return T_LEBRACE;}
"}"                     {printf("Right Brace \n"); return T_RIBRACE;}
">>"                    {printf("inp \n"); return T_INP;}
"<<"                    {printf("out \n"); return T_OUT;}
"'"                     {printf("Tone \n"); return T_TONE;}
"%"                     {printf("Percent \n"); return T_PERCENT;}
```

```
">"                     {printf("Greater \n"); return T_GREATER;}
"<"                     {printf("Smaller \n"); return T_SMALLER;}

%Comments
"/*"                    {printf("Comment \n"); BEGIN(C_COMMENT); }
<C_COMMENT>"*/"         {printf("End comment \n"); BEGIN(INITIAL); }
<C_COMMENT>.            { }
<C_COMMENT>\n           {printf("New line in comments\n"); return T_NEWLINE; }

<<EOF>>                 {printf("!Translation Complete!\n"); return T_EOF;}
.                       { }

%%
```

3. ***Τελικό BISON αρχείο***

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

//Extern from Flex
extern FILE *yyin;
extern int yylex();
extern int yylineno;
extern char *yytext;


void yyerror(const char *message);
int MAX_ERRORS=5;

int error_count=0;
int flag_err_type=0;
int scope=0;

%}

//TOKENS
%define parse.error verbose
```

```
%token T_PROG       "PROGRAM"
%token T_SMAIN      "STARTMAIN"
%token T_EMAIN      "ENDMAIN"
%token T_VARS       "VARS"
%token T_CHAR       "CHAR"
%token T_INTEGER    "INTEGER"
%token T_STRUCT     "STRUCT"
%token T_ENDSTRUCT  "ENDSTRUCT"
%token T_FUNCT      "FUNCTION"
%token T_ENDFUNC    "END_FUNCTION"
%token T_TYPEDEF    "TYPEDEF"
%token T_PRINT      "PRINT"
%token T_IF         "IF"
%token T_ENDIF      "ENDIF"
%token T_WHILE      "WHILE"
%token T_ENDWHILE   "ENDWHILE"
%token T_ELSE       "ELSE"
%token T_ELSEIF     "ELSEIF"
%token T_FOR        "FOR"
%token T_IN         "IN"
%token T_TO         "TO"
%token T_STEP       "STEP"
%token T_ENDFOR     "ENDFOR"
%token T_RETURN     "RETURN"
%token T_SWITCH     "SWITCH"
%token T_CASE       "CASE"
%token T_DEF        "DEFAULT"
%token T_ESWITCH    "ENDSWITCH"
%token T_BREAK      "BREAK"
%token T_THEN       "THEN"
%token T_AND        "AND"
%token T_OR         "OR"
%token T_OROP       "||"
%token T_ANDOP      "&&"
%token T_EQUAL      "=="
%token T_PLUS       "+"
%token T_MINUS      "-"
%token T_MULTIPLY   "*"
%token T_DIVIDE     "/"
%token T_POWER      "^"
%token T_NOT        "!"
%token T_LEPARENTH  "("
%token T_RIPARENTH  ")"
%token T_SEMICOL    ";"
%token T_DOT        "."
%token T_COMMA      ","
%token T_ASSIGN     "="
```

```
%token T_METH       "::"
%token T_COL        ":"
%token T_LEBRACKET  "["
%token T_RIBRACKET  "]"
%token T_REFER      "&"
%token T_LEBRACE    "{"
%token T_RIBRACE    "}"
%token T_INP        ">>"
%token T_OUT        "<<"
%token T_GREATER    ">"
%token T_SMALLER    "<"
%token T_NOTEQUAL   "!="
%token T_NEWLINE    "NEWLINE"
%token T_ID         "ID"
%token T_PERCENT    "%"
%token T_TONE       "'"
%token T_NUMBER     "NUMBER"
%token T_TAB        "TAB"
%token T_EOF 0      "End Of File"




%%
program:                        T_PROG T_ID newline varexp struct function
main T_EOF
                                ;
newline:                        T_NEWLINE
                                |T_NEWLINE newline
                                ;
struct:                         struct_type struct newline
                                |%empty
                                ;
struct_type:                    T_STRUCT T_ID newline varexp T_ID T_ENDSTRUCT
                                |T_TYPEDEF T_STRUCT T_ID newline varexp T_ID
T_ENDSTRUCT
                                ;
function:                       T_FUNCT T_ID T_LEPARENTH var T_RIPARENTH
newline varexp functionbody return T_ENDFUNC newline
                                |function function
                                |%empty
                                ;
varexp:                         T_VARS vartype var T_SEMICOL newline
                                |varexp varexp
                                |%empty
                                ;
vartype:                         T_CHAR
                                |T_INTEGER
```

```
                                ;
var:                             T_ID
                                |T_ID T_COMMA var
                                |%empty
                                ;
functionbody:                   command
                                |functionbody command
                                |%empty
                                ;
command:                        assignment newline
                                |loop newline
                                |check newline
                                |print newline
                                |break newline
                                ;
assignment:                     T_ID T_ASSIGN expression
                                |T_ID T_ASSIGN expression T_SEMICOL
                                ;
expression:                     literal
                                |T_ID T_LEPARENTH var T_RIPARENTH
                                |T_ID T_LEBRACKET T_NUMBER T_RIBRACKET
                                |operation
                                ;
operation:                      literal noperator literal
                                |operation noperator operation
                                |operation noperator literal
                                |literal noperator operation
                                |T_LEPARENTH operation T_RIPARENTH
                                ;
literal:                        T_NUMBER
                                |T_ID
                                ;
noperator:                       T_PLUS
                                |T_MINUS
                                |T_POWER
                                |T_MULTIPLY
                                |T_DIVIDE
                                |T_ASSIGN
                                |T_PERCENT
                                ;
loop:                            forloop
                                |whileloop
                                ;
forloop:                         T_FOR T_ID T_ASSIGN T_NUMBER T_TO T_NUMBER
T_STEP T_NUMBER newline functionbody T_ENDFOR
                                ;
whileloop:                      T_WHILE T_LEPARENTH condition T_RIPARENTH
newline functionbody T_ENDWHILE
```

```
                                ;
condition:                      literal operators literal
                                ;
operators:                       loperator
                                |coperator
                                ;
loperator:                       T_GREATER
                                |T_SMALLER
                                |T_EQUAL
                                |T_NOTEQUAL
                                ;
coperator:                       T_AND
                                |T_OR
                                ;
check:                           checkif
                                |checkcase
                                ;
checkif:                         T_IF T_LEPARENTH condition T_RIPARENTH
T_THEN newline functionbody T_ENDIF
                                |T_IF T_LEPARENTH condition T_RIPARENTH
T_THEN newline functionbody T_ELSE newline functionbody  T_ENDIF
                                |T_IF T_LEPARENTH condition T_RIPARENTH
T_THEN newline functionbody elseif T_ELSE newline functionbody  T_ENDIF
                                ;
elseif:                          T_ELSEIF newline functionbody
                                |elseif elseif
                                ;
checkcase:                       T_SWITCH T_LEPARENTH expression T_RIPARENTH
newline case default T_ESWITCH
                                ;
case:                            T_CASE T_LEPARENTH expression T_RIPARENTH
T_COL newline functionbody
                                |case case
                                ;
default:                         T_DEF T_COL newline functionbody
                                |%empty
                                ;
print:                           T_PRINT T_LEPARENTH T_TONE message T_TONE
T_COMMA T_LEBRACKET var T_RIBRACKET T_RIPARENTH T_SEMICOL
                                |T_PRINT T_LEPARENTH T_TONE message T_TONE
T_RIPARENTH T_SEMICOL
                                ;
message:                         literal
                                |literal message
                                |%empty
                                ;
break:                           T_BREAK T_SEMICOL
                                ;
```

```
return:                         T_RETURN literal T_SEMICOL newline
                                ;
main:                           T_SMAIN newline varexp functionbody T_EMAIN
                                ;

%%
int main(int argc, char* argv[]){
 int token;
    if(argc>1){
        yyin=fopen(argv[1], "r");
        if(yyin==NULL)
        {
            perror("Error opening file");
            return -1;
        }
     }

    yyparse();
    fclose(yyin);
    return 0;
}



void yyerror(const char *message)
{
    error_count++;

    if(flag_err_type==0){
        printf("-> ERROR at line %d caused by %s : %s\n", yylineno, yytext,
message);
    }else if(flag_err_type==1){

        printf("-> ERROR at line %d %s\n", yylineno, message);
    }
    flag_err_type = 0;
    if(MAX_ERRORS <= 0) return;
    if(error_count == MAX_ERRORS){
        printf("Max errors (%d) detected.\n", MAX_ERRORS);
        exit(-1);
    }
}
```

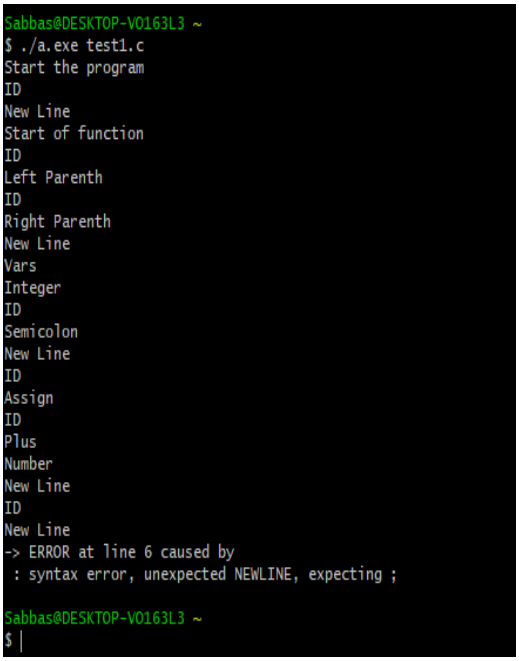
4. <u>ΣΧΟΛΙΑ</u>

- Δεν έχουμε ολοκληρώσει το 3ο ερώτημα.
- Θεωρούμε ότι οι FUNCTION/END_FUNCTION και STARTMAIN/ENDMAIN περιέχουν τουλάχιστον 1 εντολή.
- Θεωρούμε πως τα comments ξεκινούν μετα την δήλωση PROGRAM <name> .

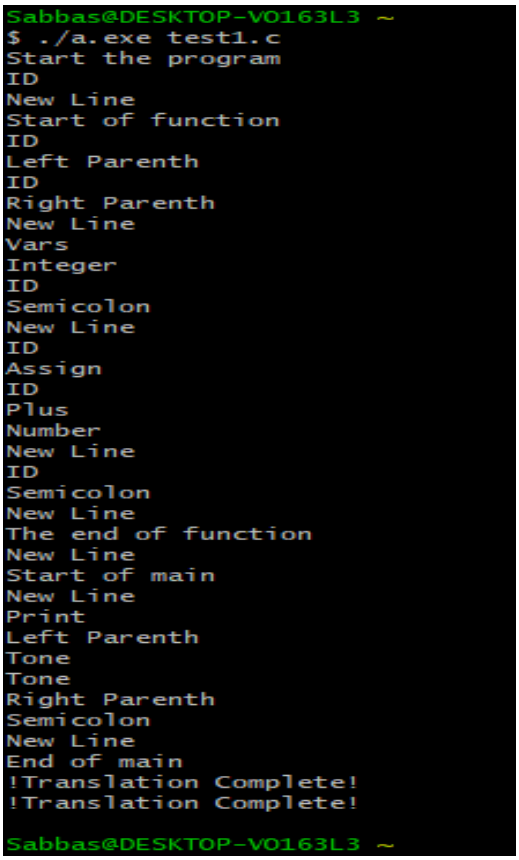
5. ***Screenshots Λειτουργίας:***

ΕΡΩΤΗΜΑ 1.b

Εδώ θέλουμε να υποδείξουμε την ικανότητα του κώδικά μας να εμφανίζει τα συντακτικά λάθη (κατά το πρότυπο της εκφώνησης). Οπότε έχουμε δημιουργήσει ένα πολύ απλό αρχείο κώδικα(test1.c) απλά για να δείξουμε αυτή την ιδιότητα.



Στο test1.c έχουμε επιτηδευμένο λάθος κι όπως φαίνεται το σφάλμα αναγνωρίζεται και επισημαίνεται με χαρακτηριστική ακρίβεια (γραμμή όπου συνέβη , λεπτομέρειές του)



Μετά την διόρθωση του test1.c έχουμε τη σωστή μεταγλώττιση του κώδικα

ΕΡΩΤΗΜΑ 2

Στο ερώτημα αυτό δίνουμε τη δυνατότητα στον αναλυτή μας να αναγνωρίζει struct-δομές. Τα αποτελέσματα διαφαίνονται παρακάτω:

```
Sabbas@DESKTOP-VO163L3 ~
$ ./a.exe test2.c
Start the program
ID
New Line
This is a struct
ID
New Line
Vars
Character
ID
Comma
ID
Semicolon
New Line
Vars
Integer
ID
Comma
ID
Semicolon
New Line
ID
End of struct
New Line
Start of function
ID
Left Parenth
ID
Right Parenth
New Line
Vars
Integer
ID
Semicolon
New Line
ID
Assign
Number
Semicolon
New Line
ID
Semicolon
New Line
The end of function
New Line
Start of main
New Line
Vars
Character
ID
Semicolon
New Line
Print
Left Parenth
Tone
ID
Tone
Comma
Left Bracket
ID
Right Bracket
Right Parenth
Semicolon
New Line
End of main
!Translation Complete!
!Translation Complete!
```

Εδώ βλέπουμε την επιτυχή εκτέλεση του test2.c, με χρήση struct αμέσως μετά την αρχή του προγράμματος .

```
Sabbas@DESKTOP-VO163L3 ~
$ ./a.exe test2.c
Start the program
ID
New Line
This is a struct
New Line
-> ERROR at line 3 caused by
 : syntax error, unexpected NEWLINE, expecting ID

Sabbas@DESKTOP-VO163L3 ~
$
```

Απεναντίας, βλέπουμε ανεπιτυχή μεταγλώττιση, μετά τον έλεγχο της λανθασμένης struct.

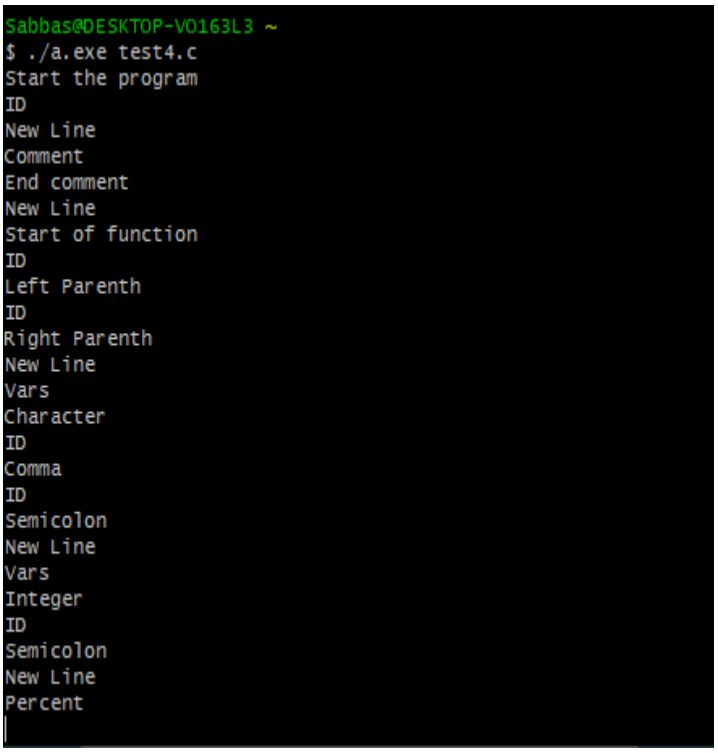
ΕΡΩΤΗΜΑ 4

Στο ερώτημα αυτό θέλουμε να παρέχεται η ικανότητα χρήσης σχολίων μεταξύ των εντολών στον κωδικα μας .

```
Sabbas@DESKTOP-VO163L3 ~
$ ./a.exe test4.c
Start the program
ID
New Line
Comment
End comment
New Line
Start of function
ID
Left Parenth
ID
Right Parenth
New Line
Vars
Character
ID
Comma
ID
Semicolon
New Line
Vars
Integer
ID
Semicolon
New Line
Comment
New line in comments
New line in comments
End comment
New Line
while
Left Parenth
ID
Equal
Number
Right Parenth
New Line
Print
Left Parenth
Tone
ID
Tone
Comma
Left Bracket
ID
Right Bracket
Right Parenth
Semicolon
New Line
End of while
New Line
Returning...
Number
Semicolon
New Line
The end of function
New Line
Comment
New line in comments
New line in comments
New line in comments
End comment
New Line
Start of main
New Line
If
Left Parenth
ID
Equal
ID
Right Parenth
Then
New Line
Print
Left Parenth
Tone
ID
Tone
Right Parenth
Semicolon
New Line
End of if
New Line
End of main
!Translation Complete!
!Translation Complete!
```

Το screenshot με διάφορα comment του ζητούμενου τύπου .

ΠΡΟΥΠΟΘΕΣΗ!!!
Να μην ξεκινάμε με comment πριν γράψουμε <PROGRAM name>



Αλλάξαμε ένα comment και το μετατρέψαμε σε μορφή %comment%

Παρατηρούμε πως το πρόγραμμα "κολλάει" και δεν συνεχίζει

Ένα συνολικο test αρχείο με όλες τις εντολές που επιτρέπονται από τους αναλυτές μας!