Πολυτεχνική Σχολή

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Αρχές Γλωσσών Προγραμματισμού & Μεταφραστών 2020-2021

Εαρινό Εξάμηνο 2021

Εργαστηριακή άσκηση

1. Περιεχόμενα

1. Περιεχόμενα **2**

2. Σύσταση Ομάδας **3**

3. Περιγραφή Της Γραμματικής Της Γλώσσας σε BNF **3**

4. Τελικό Flex Αρχείο **6**

5. Τελικό BISON *Αρχείο* **9**

6. Σχόλια / Παραδοχές: **16**

7. Screenshots Λειτουργίας: **17**

1ο Ερώτημα 17

2ο Ερώτημα 18

4ο Ερώτημα 19

2. Σύσταση Ομάδας

Γκέκα Βασιλική Λευκοθέα	ΑΜ 1059697	4ο έτος	up1059697@upnet.gr
Ζαφειρέλης Αντώνιος Ζαφείριος	ΑΜ 1059605	4ο έτος	up1059605@g.upatras.gr
Ζύμνης Μάρκελλος	ΑΜ 1059562	4ο έτος	st1059562@ceid.upatras.gr

3. Περιγραφή Της Γραμματικής Της Γλώσσας σε BNF

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

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

<newline> :         [\r\n]

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

<struct_type> :     STRUCT <id>  <newline> <varksm> <id>  ENDSTRUCT
                    |TYPEDEF STRUCT <id>  <newline> <varksm> <id>  ENDSTRUCT

<function>  :       FUNCTION <id>  ( <var> )  <newline>   <varksm>   <functionbody>   <return>
                    ENDFUNCTION <newline>
                    |<function> <function>
                    |ε

<varksm> :          VARS <vartype> <var> ; <newline>
                    |<varksm> <varksm>
                    |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>
                    |<comment> <newline>
                    |<mcomment> <newline>
```

```
<assignment> :          <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>

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

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

<loop> :                 <forloop>
                        |<whileloop>

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

<whileloop> :           WHILE ( <condition> ) <newline> <functionbody> 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> ENDSWITCH

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

<default> :               DEFAULT : <newline> <functionbody>
                          |ε
<print> :                 PRINT (" <message> " , \[ <var> \] ) ;

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

<break> :                 BREAK ;

<comment> :               % <message>

<mcomment> :              /* <messages> */

<messages> :              < message >
                          |<message> <newline> <messages>

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

<main> :                  STARTMAIN <newline> <varksm> <functionbody> ENDMAIN
```

4. Τελικό Flex Αρχείο

```
%{

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "main.tab.h"

%}

%option         noyywrap
%option         yylineno

LETTER          [a-zA-Z]
ID              [a-z][a-z0-9]*
DIGIT           [0-9]
NUMBER          [0-9][0-9]*
NEWLINE         \r\n
TAB             [\t]
other_chars     [.\-_]

%%

"PROGRAM"       {printf("Program Started\n"); return T_PROG;}
"VARS"          {printf("Variable Declaration Started\n"); return
T_VARS;}
"CHAR"          {printf("Found Character Variable\n"); return T_CHAR;}
"INTEGER"       {printf("Found Integer variable\n"); return T_INT;}
"STRUCT"        {printf("Found Struct Declaration\n"); return T_STRCT;}
"ENDSTRUCT"     {printf("Struct Declaration Ended\n"); return T_ESTRCT;}
"TYPEDEF"       {printf("Type Defintion Found\n"); return T_TDEF;}
"PRINT"         {printf("Print Statement Found\n"); return T_PRINT;}
"IF"            {printf("If Statement Initiated\n"); return T_IF;}
"THEN"          {printf("Then Statement Found \n"); return T_THEN;}
"ENDIF"         {printf("If Statement Ended\n"); return T_ENDIF;}
"WHILE"         {printf("Found While\n"); return T_WHILE;}
"ENDWHILE"      {printf("While Condition Ended\n"); return T_EWHILE;}
"ELSE"          {printf("Found Else\n"); return T_ELSE;}
"ELSEIF"        {printf("Found ElseIf\n"); return T_ELSEIF;}
"FOR"           {printf("For Statement Found\n"); return T_FOR;}
```

```
"TO"            {printf("For Condition TO\n"); return T_TO;}
"STEP"          {printf("For Condition STEP\n"); return T_STEP;}
"ENDFOR"        {printf("For Condition Ended\n"); return T_ENDFOR;}
"STARTMAIN"     {printf("Main Function Started\n"); return T_SMAIN;}
"ENDMAIN"       {printf("Main Function Ended\n"); return T_EMAIN;}
"FUNCTION"      {printf("Function Declaration\n"); return T_FUNCT;}
"END_FUNCTION"  {printf("Function End Reached\n"); return T_EFUNCT;}
"RETURN"        {printf("Value Return Found\n"); return T_RETURN;}
"SWITCH"        {printf("Found A Switch\n"); return T_SWITCH;}
"CASE"          {printf("Switch Case Found\n"); return T_CASE;}
"DEFAULT"       {printf("Switch Default\n"); return T_DEF;}
"ENDSWITCH"     {printf("Switch Ended\n"); return T_ESWITCH;}
"BREAK"         {printf("Break Found\n"); return T_BREAK;}
"AND"           {printf("Logical And Statemnt Found\n"); return T_AND;}
"OR"            {printf("Logical Or Statement Found\n"); return T_OR;}


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

"="             {printf("Assign\n"); return T_ASSIGN;}
">"             {printf("Greater Than\n"); return T_GREATER;}
"<"             {printf("Smaller Than\n"); return T_SMALLER;}
"=="            {printf("Equal To\n"); return T_EQUAL;}
"!="            {printf("Not Equal To\n"); return T_NEQUAL;}
"+"             {printf("Plus Sign\n"); return T_PLUS;}
"-"             {printf("Minus Sign\n"); return T_MINUS;}
"^"             {printf("Power Sign\n"); return T_POWER;}
"*"             {printf("Multiplication Sign\n"); return T_MULT;}
"/"             {printf("Division Sign\n"); return T_DIV;}
"{"             {printf("Left Brace\n"); return T_LBRACE;}
"}"             {printf("Right Brace\n"); return T_RBRACE;}
"["             {printf("Left Bracket\n"); return T_LBRACKET;}
"]"             {printf("Right Bracket\n"); return T_RBRACKET;}
"("             {printf("Left Parenthesis\n"); return T_LPARENTH;}
")"             {printf("Right Parenthesis\n"); return T_RPARENTH;}
";"             {printf("Semicolon\n"); return T_SEMICOL;}
":"             {printf("Column\n"); return T_COL;}
"."             {printf("Dot\n"); return T_DOT;}
","             {printf("Comma\n"); return T_COMMA;}
"\""            {printf("Tonos\n"); return T_TONOS;}
```

```
" "                {   }
"%"                {printf("Line Comment"); return T_COMMENT;}


<<EOF>>            return T_EOF;

.                  {printf("Unknown\n");}

%%
```

5. Τελικό BISON Αρχείο

```
%{

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

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;


%}

%define parse.error verbose



%token T_PROG       "PROGRAM"
%token T_VARS       "VARS"
%token T_CHAR       "CHAR"
%token T_INT        "INTEGER"
%token T_STRCT      "STRUCT"
%token T_ESTRCT     "END STRUCT"
%token T_TDEF       "TYPE DEF"
%token T_PRINT      "PRINT"
%token T_IF         "IF"
%token T_ENDIF      "ENDIF"
%token T_WHILE      "WHILE"
%token T_EWHILE     "END WHILE"
%token T_ELSE       "ELSE"
%token T_ELSEIF     "ELSE IF"
```

```
%token T_FOR          "FOR"
%token T_TO           "TO"
%token T_STEP         "STEP"
%token T_ENDFOR       "END FOR"
%token T_SMAIN        "START MAIN"
%token T_EMAIN        "END MAIN"
%token T_FUNCT        "FUNCTION"
%token T_EFUNCT       "END FUNCTION"
%token T_RETURN       "RETURN"
%token T_SWITCH       "SWITCH"
%token T_CASE         "CASE"
%token T_DEF          "DEFAULT"
%token T_ESWITCH      "END SWITCH"
%token T_BREAK        "BREAK"
%token T_AND          "AND"
%token T_OR           "OR"
%token T_GREATER      ">"
%token T_SMALLER      "<"
%token T_EQUAL        "=="
%token T_NEQUAL       "!="
%token T_PLUS         "+"
%token T_MINUS        "-"
%token T_POWER        "^"
%token T_MULT         "*"
%token T_DIV          "/"
%token T_LBRACE       "{"
%token T_RBRACE       "}"
%token T_LBRACKET     "["
%token T_RBRACKET     "]"
%token T_LPARENTH     "("
%token T_RPARENTH     ")"
%token T_SEMICOL      ";"
%token T_COL          ":"
%token T_DOT          "."
%token T_COMMA        ","
%token T_TONOS        "\""
%token T_NLINE        "NEW LINE"
%token T_COMMENT      "%"
%token T_THEN         "THEN"
%token T_ASSIGN       "="
%token T_ID           "ID"
%token T_NUMBER       "NUMBER"
%token T_EOF     0    "EOF"
```

```
%%


program:                        T_PROG T_ID T_NLINE varksm struct
function main T_EOF
                                ;

struct:                         struct_type struct T_NLINE
                                |%empty
                                ;

struct_type:                    T_STRCT T_ID T_NLINE varksm T_ID T_ESTRCT
                                |T_TDEF T_STRCT T_ID T_NLINE varksm T_ID
T_ESTRCT
                                ;


function:                       T_FUNCT T_ID T_LPARENTH var T_RPARENTH
T_NLINE varksm functionbody return T_EFUNCT T_NLINE
                                |function function
                                |%empty
                                ;

varksm:                         T_VARS vartype var T_SEMICOL T_NLINE
                                |varksm varksm
                                |%empty
                                ;

vartype:                        T_CHAR
                                |T_INT
                                ;

var:                            T_ID
                                |T_ID T_COMMA var
                                |%empty
                                ;

functionbody:                   command
```

```
                                |functionbody command
                                |%empty
                                ;

command:                        assignment T_NLINE
                                |loop T_NLINE
                                |check T_NLINE
                                |print T_NLINE
                                |break T_NLINE
                                |comment T_NLINE
                                |mcomment T_NLINE
                                ;

assignment:                     T_ID T_ASSIGN expression T_SEMICOL
                                ;

expression:                     literal
                                |T_ID T_LPARENTH var T_RPARENTH
                                |T_ID T_LBRACKET T_NUMBER T_RBRACKET
                                |operation
                                ;

operation:                      literal noperator literal
                                |operation noperator operation
                                |operation noperator literal
                                |literal noperator operation
                                |T_LPARENTH operation T_RPARENTH
                                ;


literal:                        T_NUMBER
                                |T_ID
                                ;

noperator:                      T_PLUS
                                |T_MINUS
                                |T_POWER
                                |T_MULT
                                |T_DIV
                                ;

loop:                           forloop
                                |whileloop
                                ;

forloop:                        T_FOR T_ID T_ASSIGN T_NUMBER T_TO
T_NUMBER T_STEP T_NUMBER T_NLINE functionbody T_ENDFOR
                                ;

whileloop:                      T_WHILE T_LPARENTH condition T_RPARENTH
T_NLINE functionbody T_EWHILE
                                ;

condition:                      literal operators literal
                                ;

operators:                       loperator
                                |coperator
                                ;

loperator:                      T_SMALLER
                                |T_GREATER
                                |T_EQUAL
                                |T_NEQUAL
                                ;

coperator:                      T_AND
                                |T_OR
                                ;

check:                          checkif
                                |checkcase
                                ;

checkif:                        T_IF T_LPARENTH condition T_RPARENTH
T_THEN T_NLINE functionbody  T_ENDIF
                                |T_IF T_LPARENTH condition T_RPARENTH
T_THEN T_NLINE functionbody T_ELSE T_NLINE functionbody  T_ENDIF
                                |T_IF T_LPARENTH condition T_RPARENTH
T_THEN T_NLINE functionbody elseif T_ELSE T_NLINE functionbody  T_ENDIF
                                ;

elseif:                         T_ELSEIF T_NLINE functionbody
                                |elseif elseif
                                ;
```

```
checkcase:                      T_SWITCH T_LPARENTH expression T_RPARENTH
T_NLINE case default T_ESWITCH
                                ;

case:                           T_CASE T_LPARENTH expression T_RPARENTH
T_COL T_NLINE functionbody
                                |case case
                                ;

default:                        T_DEF T_COL T_NLINE functionbody
                                |%empty
                                ;

print:                          T_PRINT T_LPARENTH T_TONOS message
T_TONOS T_COMMA T_LBRACKET var T_RBRACKET T_RPARENTH T_SEMICOL
                                ;

message:                        literal
                                |literal message
                                |%empty
                                ;

break:                          T_BREAK T_SEMICOL
                                ;

comment:                        T_COMMENT message
                                ;

mcomment:                       T_DIV T_MULT messages T_MULT T_DIV
                                ;

messages:                       message
                                |message T_NLINE messages
                                ;

return:                         T_RETURN literal T_SEMICOL T_NLINE
                                ;

main:                           T_SMAIN T_NLINE varksm functionbody
T_EMAIN
                                ;
```

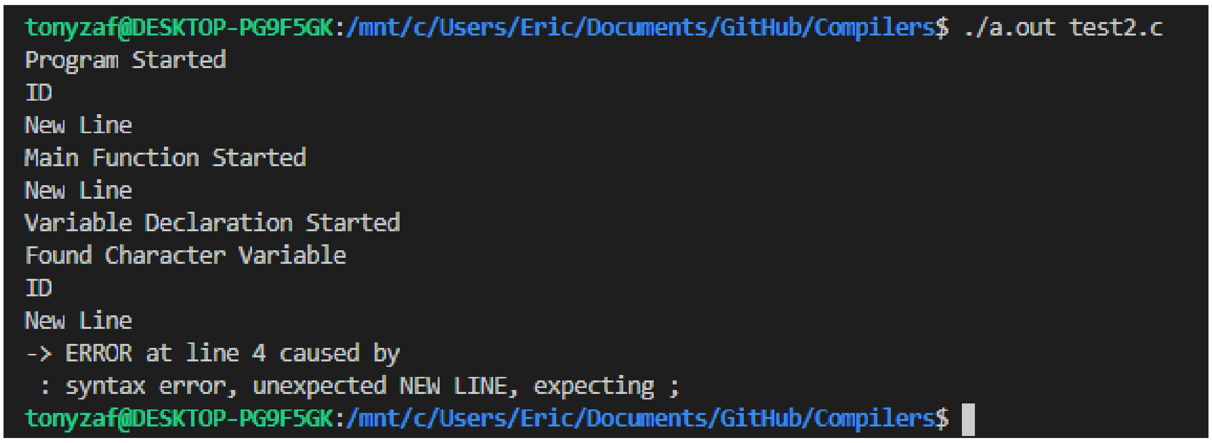
```
%%


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);
    }
}
```

6. Σχόλια / Παραδοχές:

- Θεωρήσαμε Τα Σχόλια Σαν Εντολές.
- Θεωρήσαμε ότι τα σχόλια ξεκινούν απο την αρχή της γραμμής.
- Θεωρήσαμε ότι η δήλωση συνάρτησης FUNCTION/MAIN περιέχει τουλάχιστον μία εντολή.

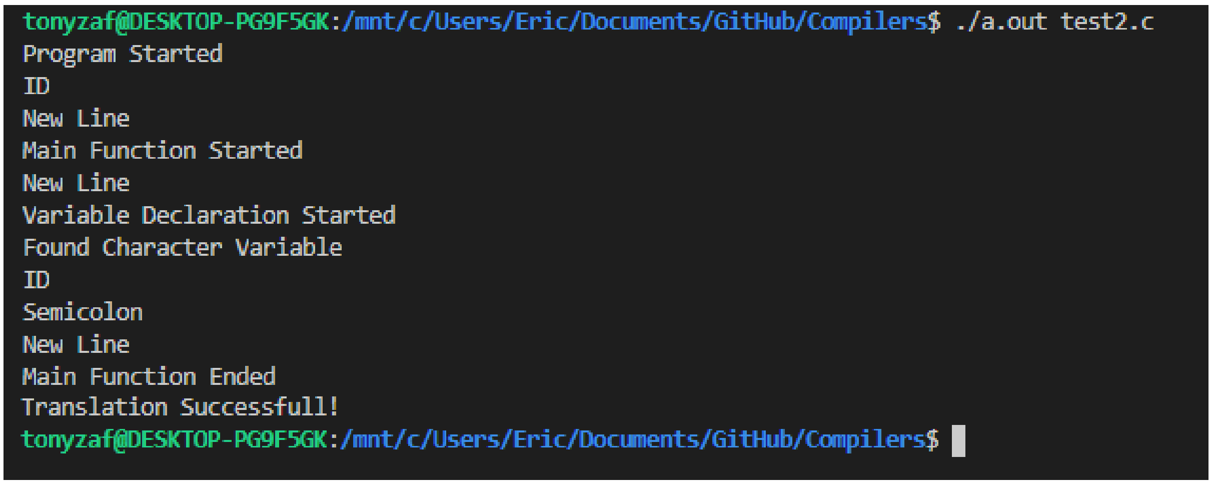
7. Screenshots Λειτουργίας:

1ο Ερώτημα



```
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$ ./a.out test2.c
Program Started
ID
New Line
Main Function Started
New Line
Variable Declaration Started
Found Character Variable
ID
New Line
-> ERROR at line 4 caused by
 : syntax error, unexpected NEW LINE, expecting ;
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$
```

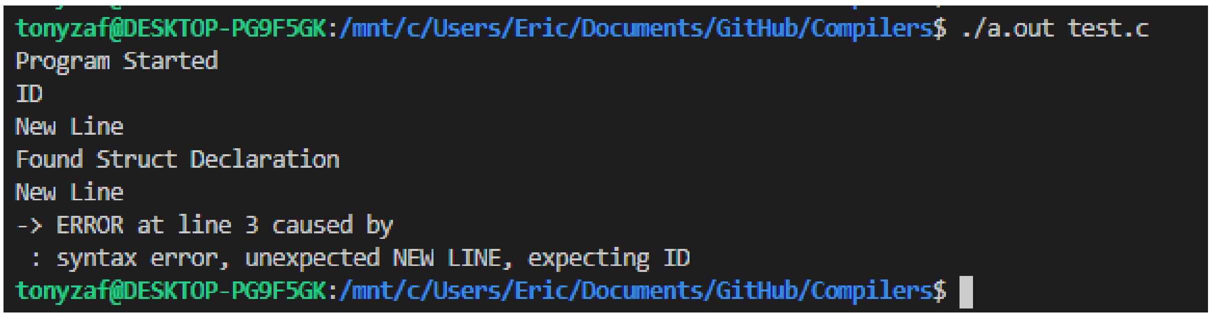
Ανεπιτυχής Έλεγχος με Εμφάνιση Error Message



```
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$ ./a.out test2.c
Program Started
ID
New Line
Main Function Started
New Line
Variable Declaration Started
Found Character Variable
ID
Semicolon
New Line
Main Function Ended
Translation Successfull!
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$
```

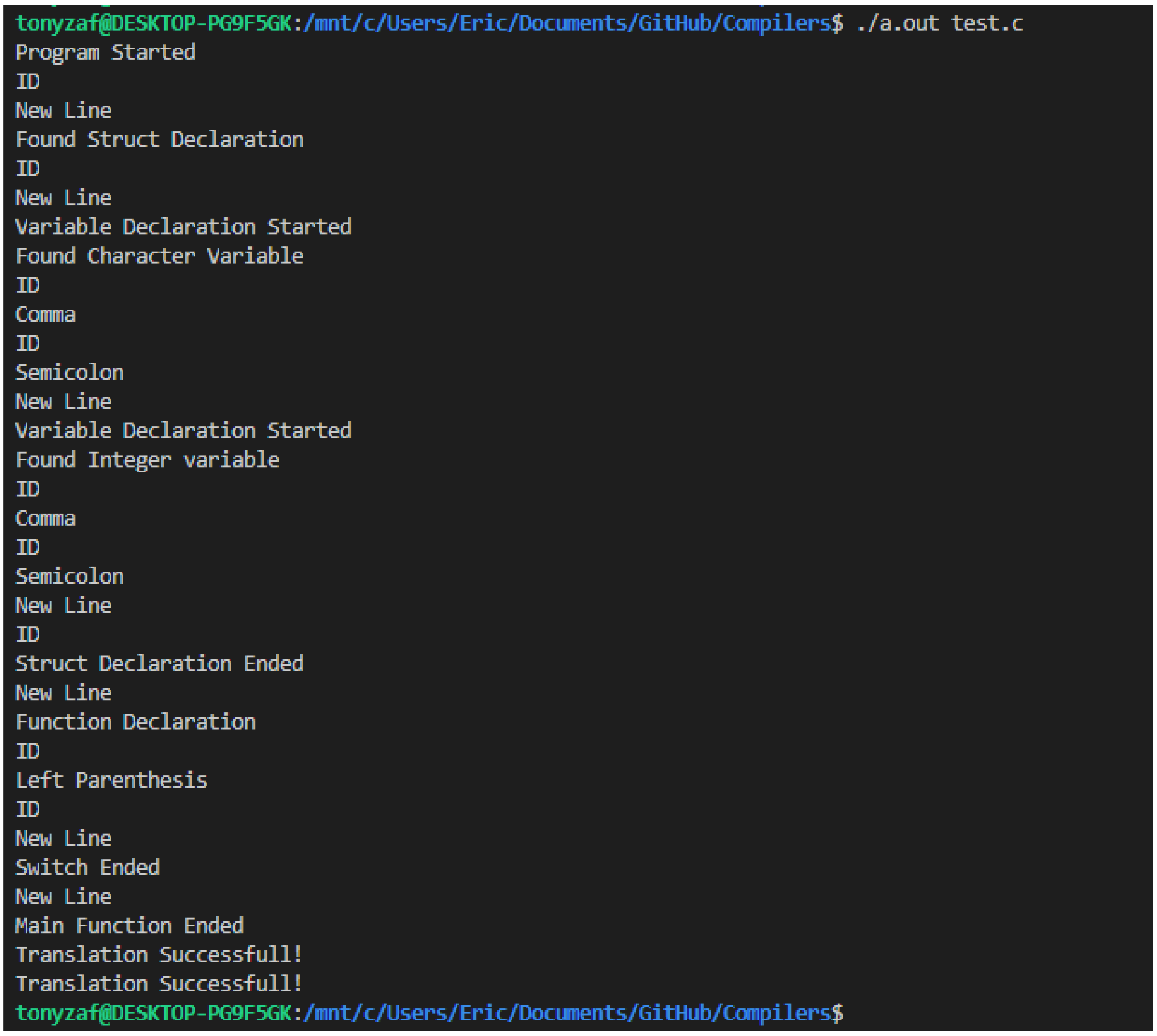
Επιτυχής Έλεγχος Με Εμφάνιση Σχετικού Μηνύματος

2ο Ερώτημα



```
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$ ./a.out test.c
Program Started
ID
New Line
Found Struct Declaration
New Line
-> ERROR at line 3 caused by
 : syntax error, unexpected NEW LINE, expecting ID
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$
```

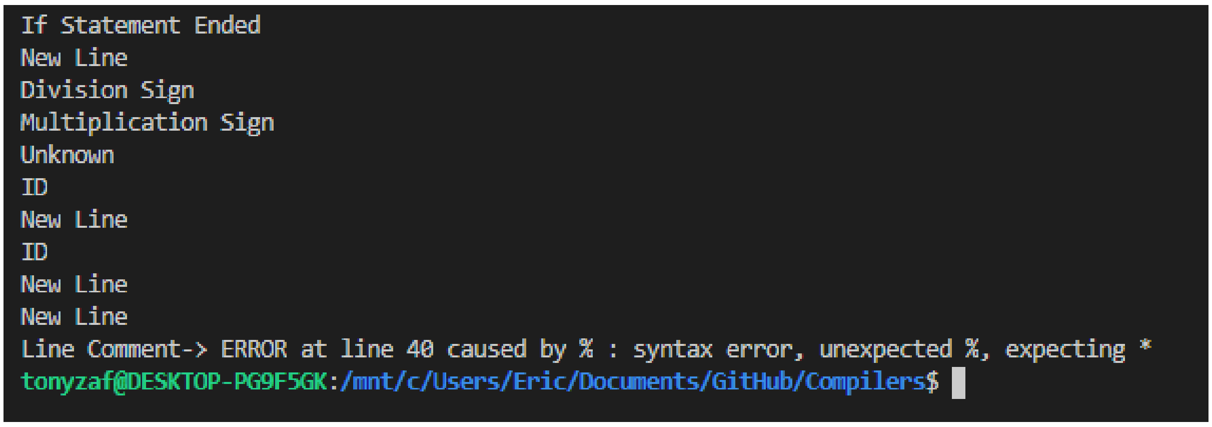
Ανεπιτυχής Έλεγχος με Εμφάνιση Error Message



```
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$ ./a.out test.c
Program Started
ID
New Line
Found Struct Declaration
ID
New Line
Variable Declaration Started
Found Character Variable
ID
Comma
ID
Semicolon
New Line
Variable Declaration Started
Found Integer variable
ID
Comma
ID
Semicolon
New Line
ID
Struct Declaration Ended
New Line
Function Declaration
ID
Left Parenthesis
ID
New Line
Switch Ended
New Line
Main Function Ended
Translation Successfull!
Translation Successfull!
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$
```

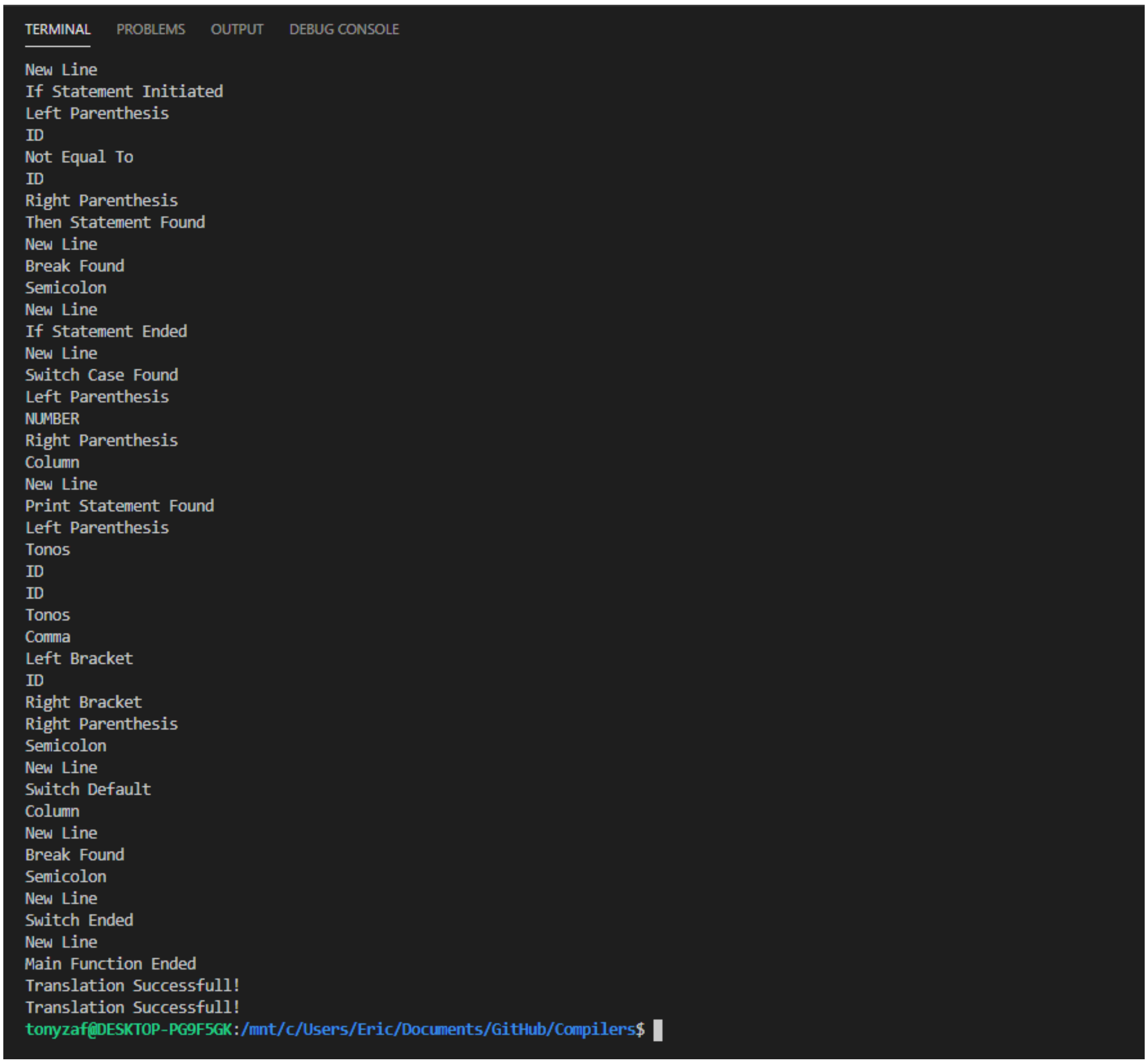
Επιτυχής Έλεγχος Με Εμφάνιση Σχετικού Μηνύματος

4ο Ερώτημα



```
If Statement Ended
New Line
Division Sign
Multiplication Sign
Unknown
ID
New Line
ID
New Line
New Line
Line Comment-> ERROR at line 40 caused by % : syntax error, unexpected %, expecting *
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$
```

Ανεπιτυχής Έλεγχος με Εμφάνιση Error Message



```
New Line
If Statement Initiated
Left Parenthesis
ID
Not Equal To
ID
Right Parenthesis
Then Statement Found
New Line
Break Found
Semicolon
New Line
If Statement Ended
New Line
Switch Case Found
Left Parenthesis
NUMBER
Right Parenthesis
Column
New Line
Print Statement Found
Left Parenthesis
Tonos
ID
ID
Tonos
Comma
Left Bracket
ID
Right Bracket
Right Parenthesis
Semicolon
New Line
Switch Default
Column
New Line
Break Found
Semicolon
New Line
Switch Ended
New Line
Main Function Ended
Translation Successfull!
Translation Successfull!
tonyzaf@DESKTOP-PG9F5GK:/mnt/c/Users/Eric/Documents/GitHub/Compilers$
```

Επιτυχής Έλεγχος Με Εμφάνιση Σχετικού Μηνύματος