ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

Εργασία Flex & Bison

ΟΝΟΜΑΤΕΠΩΝΥΜΑ / ΑΜ:
Φωτοπούλου Μαρία-Γεωργία, 1059597
Βουλδής Άγγελος, 1059624
Κωνσταντίνος Μωραγέμος, 1059583

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
Τμήμα Μηχανικών Ηλεκτρονικών
Υπολογιστών και Πληροφορικής

ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ

Εργασία Flex & Bison

Περιεχόμενα

ΕΙΣΑΓΩΓΗ

Στην εργασία αυτή ασχοληθήκαμε με την λεκτική και συντακτική ανάλυση μιας ψευδογλώσσας που ακολουθεί την λογική της γλώσσας C.

Χρησιμοποιώντας τον συντακτικό αναλυτή Bison και τον λεκτικό αναλυτή Flex υλοποιήσαμε τα παρακάτω:

- Ορισμός συνάρτησης και κλήση της
- Αρχικοποίηση μεταβλητών
- Υποστήριξη σχολίων
- Εντολές βρόγχου και συνθήκης (If , For loop , While, Switch/Case)
- Υποστήριξη εμφάνισης μηνυμάτων
- Υλοποίησης των operators +, -, *, /
- Υλοποίηση του κύριου προγράμματος της Main

Δημιουργήσαμε συνολικά 4 αρχεία τα οποία είναι ο λεξικός αναλυτής (lexer), ο συντακτικός αναλυτής (parser), ένα αρχείο c και ένα αρχείο .cme που αποτελεί το αρχείο της ψευδογλώσσας μας.

Ο **λεξικός** αναλυτής (αρχείο lexer.l) περιέχει το αλφάβητο της γλώσσας μας, δηλαδή καθορίζουμε στο πρόγραμμα μας τι να αναγνωρίζει η γλώσσα μας και τί όχι. Διαχωρίζει τους χαρακτήρες του προγράμματος σε ομάδες. Οι λέξεις που αναγνωρίζει ο λεξικός μας αναλυτής λέγονται και tokens.

Ο **συντακτικός** αναλυτής (parser.y) καθορίζει την σωστή δομή που θα έχει η γλώσσα που ορίσαμε στο λεξικό αναλυτή για να υπακούει στην γλώσσα που θέλουμε να φτιάξουμε.

Το αρχείο της ψευδογλώσσας της C **(.cme)** περιέχει κώδικα γραμμένο με τους κανόνες που δώθηκαν στην εκφώνηση της άσκησης το οποίο είναι και το αρχείο εισόδου στον συντακτικό αναλυτή. Ο κώδικας ο οποίος καλύπτει όλες τις παραπάνω λειτουργίες που προαναφέραμε. Είναι το αρχείο το οποίο μας επιβεβαιώνει την ορθότητα της ανάλυσης μας.

Περιγραφή γραμματικής της γλώσσας σε BNF

Αρχείο BNF

```
<PROGRAM> ::= <PROGRAM><word>NEWLINE | <PROGRAM><line> | <PROGRAM><main_func>

<char> ::= [a-z]|[A-Z]
<nonzero> ::= 1|2|3|4|5|6|7|8|9
<digit> ::= 0|<nonzero>
<digits> ::= <digit>|<digit><digits>
<integers> ::= <digit>|<nonzero><digits>

<empty>   ::=
<word> ::= CHAR|<word>CHAR

<line> ::= <if_stmt> | <elseif_stmt> | <else_stmt> |<for_statement> | <function> NEW
LINE
            | <function_call>
            | <comments> NEWLINE
            | <print> NEWLINE
            | <break>
            | <variable>
            | <switch>
            | <while>
            | <import_statement> NEWLINE
            | <dictionaries> NEWLINE
            | <calc_assigmnet> NEWLINE
            | <main_func>

<break>::= BREAK QM NEWLINE
<data_type> ::= CHAR | INTEGER | IDENTIFIER | FLOAT | STRING

<variable>::= VARS  <data_type> <inspector>
        | VARS  <data_type> IDENTIFIER COMMA IDENTIFIER
        | VARS  <variable_dictionary>
        | <variable> <variable_dictionary>
        | <variable> QM
        | <variable>  <variable_dictionary> QM

<variable_dictionary>::= <data_type> <inspector>
                      | <data_type> IDENTIFIER COMMA IDENTIFIER
                      | COMMA <array>
                      | <array>
                      | COMMA IDENTIFIER
                      | IDENTIFIER COMMA IDENTIFIER
                      | <variable_dictionary> COMMA IDENTIFIER
                      | <variable_dictionary> COMMA <array>
                      | <line>

<return>::= RETURN INTEGER QM NEWLINE
```

```
        | RETURN IDENTIFIER QM NEWLINE
        | RETURN <int_op> QM NEWLINE

<if_stmt> ::= IF L_PAR <inspector> R_PAR THEN NEWLINE
        | <if_stmt> <line>
        | <if_stmt> <end_if_stmt>
        | <if_stmt> <elseif_stmt>
        | <if_stmt> <elseif_stmt> <else_stmt>

<elseif_stmt> ::= ELSEIF L_PAR <inspector> R_PAR  <line>
<else_stmt> ::= ELSE  <line>
<end_if_stmt>: ENDIF NEWLINE


<for_statement> ::= FOR IDENTIFIER COLON ASSIGN INTEGER TO INTEGER STEP INTEGER NEWL
INE
               |<for_statement> <line>
               |<for_statement> <end_for_statement>

<end_for_statement>::= ENDFOR NEWLINE

<switch> ::= SWITCH L_PAR LT IDENTIFIER GT R_PAR NEWLINE
        |SWITCH L_PAR LT IDENTIFIER GT R_PAR COMMENT
        |<switch> <case>
        |<switch> <case> <end_switch>
<case> ::= CASE L_PAR LT INTEGER GT R_PAR NEWLINE <line> <break>
<end_switch> ::= ENDSWITCH NEWLINE

<while> ::=  WHILE L_PAR <inspector_gen> R_PAR NEWLINE
        |<while> <line>
        |<while> <end_while>

<end_while> ::=  ENDWHILE NEWLINE

<array> ::= IDENTIFIER L_BRACK INTEGER R_BRACK
        | IDENTIFIER L_BRACK IDENTIFIER R_BRACK

<array_value> ::= <array> ASSIGN INTEGER

<operators> ::= EQ_OP | GE_OP | LE_OP | NE_OP | DEC_OP | INC_OP | LT | GT | AND_OP |
 OR_OP

<optional_parameters> ::=  IDENTIFIER
                     | <data_type> IDENTIFIER COMMA <data_type> IDENTIFIER
                     | IDENTIFIER IDENTIFIER COMMA <optional_parameters>
                     | IDENTIFIER IDENTIFIER

<function> ::= FUNCTION IDENTIFIER L_PAR <optional_parameters> R_PAR NEWLINE <line>
|  <function> <return> <end_function>
```

```
<end_function> ::=  END_FUNCTION NEWLINE


<function_call> ::= IDENTIFIER L_PAR <optional_parameters> R_PAR
                      | IDENTIFIER L_PAR <data_type> R_PAR
                      | IDENTIFIER L_PAR <data_type> COMMA <data_type> R_PAR
                      | IDENTIFIER L_PAR <data_type> COMMA <data_type> COMMA <data_ty
pe> R_PAR
                      | <function_call> QM NEWLINE

<inspector>::= IDENTIFIER <operators> <data_type>
           |<data_type> <operators> IDENTIFIER


<inspector_gen> ::= <inspector>
                | <inspector> <operators>
                | <inspector> <operators> <inspector>
                | <inspector_gen> QM

<comments> ::= COMMENT

<print> ::= PRINT L_PAR <data_type> R_PAR QM | PRINT L_PAR <data_type> <print_name_v
ar> R_PAR QM | <print> NEWLINE
<print_name_var> ::= L_BRACK COMMA IDENTIFIER R_BRACK
                 | L_BRACK COMMA <array> R_BRACK


<main_func> ::=  STARTMAIN  NEWLINE
           | <main_func> <line>
           | <main_func> <end_main>

<end_main> ::= ENDMAIN NEWLINE

<import_statement> ::=FROM IDENTIFIER IMPORT IDENTIFIER AS IDENTIFIER
                   | FROM IDENTIFIER IMPORT IDENTIFIER
                   | FROM IDENTIFIER IMPORT MUL
                   | IMPORT IDENTIFIER AS IDENTIFIER
                   | IMPORT IDENTIFIER
                   | FROM IDENTIFIER IMPORT IDENTIFIER COMMA IDENTIFIER

<dictionaries> ::= IDENTIFIER ASSIGN L_BRACE <dictionary_data> R_BRACE
                 | IDENTIFIER ASSIGN IDENTIFIER L_PAR L_BRACK L_PAR <dictionary_data>
 R_PAR R_BRACK R_PAR
                 | IDENTIFIER ASSIGN IDENTIFIER L_PAR <dictionary_data> <optional_par
ameters> <dictionary_data> R_PAR
                 | IDENTIFIER ASSIGN <function_call>
                 | <array_value> QM
```

```
<dictionary_data> ::= <data_type> COMMA <data_type> <optional_parameters>
                  | IDENTIFIER ASSIGN <data_type> QM
                  |/* empty */

 <term> ::= <data_type>

 <calc_assigmnet> ::= IDENTIFIER ASSIGN <int_op>

 <int_op> ::= <int_data> | <int_op> PLUS <int_data> | <int_op> MINUS <int_data>
          | <int_op> MUL <int_data>
          | <int_op> DIV <int_data>

 <int_data> ::= INTEGER | IDENTIFIER
```

Ολοκληρωμένος Κώδικας

Αρχείο Lexer.l

```
%option noyywrap

%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <math.h>
    #include <unistd.h>
    #include "parser.tab.h"

    extern FILE *yyin;
    extern FILE *yyout;
    int line_no = 1; //program's line number
    int line_init=-1; // For multiline comments & strings

    //the function of lexer analysis. Return the token
    int yylex();
    //error function
    void yyerror();
    //print statement function
    void print_return(char *token);

%}

%x ML_COMMENT

alphabet        [a-zA-Z]
alphanumeric    {alphabet}|{integer}
print           [ -~]
underscore      _
identifier      ({alphabet}|{underscore})+({alphanumeric}|{underscore})*
integer         [0-9][0-9]*
float_number    "0"|{integer}*"."{integer}+
char            \'{print}\'
string          \".*\"
WHITESPACE        [ \t]*
NEWLINE           [\n]*

%%



"%".*\n          {  print_return("COMMENT"); return COMMENT; }
```

```
<INITIAL>"/*"                   { BEGIN(ML_COMMENT); }
<ML_COMMENT>"*"+"/"             { BEGIN(INITIAL);}

<ML_COMMENT>([^*]|\n)+|.



<ML_COMMENT><<EOF>>          {yyerror("Unterminated comment", 1); return 0;}



"PROGRAM"        { print_return("PROGRAM"); return PROGRAM; }

"BREAK"          {  print_return("BREAK"); return BREAK; }
"VARS"           {  print_return("VARS"); return VARS; }

"STARTMAIN"      {  print_return("STARTMAIN");  return STARTMAIN; }
"ENDMAIN"        {  print_return("ENDMAIN"); return ENDMAIN;}

"IF"             {  print_return("IF");  return IF; }
"THEN"           {  print_return("THEN"); return THEN;}
"ELSEIF"         {  print_return("ELSEIF"); return ELSEIF; }
"ELSE"           {  print_return("ELSE"); return ELSE; }
"ENDIF"          {  print_return("ENDIF"); return ENDIF; }

"FOR"            {  print_return("FOR"); return FOR; }
"TO"             {  print_return("TO"); return TO; }
"STEP"           {  print_return("STEP"); return STEP; }
"ENDFOR"         {  print_return("ENDFOR"); return ENDFOR; }

"SWITCH"         {  print_return("SWITCH"); return SWITCH; }
"CASE"           {  print_return("CASE"); return CASE; }
"ENDSWITCH"      {  print_return("ENDSWITCH"); return ENDSWITCH; }

"RETURN"         {  print_return("RETURN"); return RETURN; }

"FUNCTION"       {  print_return("FUNCTION"); return FUNCTION; }
"END_FUNCTION"    {  print_return("END_FUNCTION"); return END_FUNCTION; }

"PRINT"          {  print_return("PRINT"); return PRINT; }

"WHILE"          {  print_return("WHILE"); return WHILE;}
"ENDWHILE"       {  print_return("ENDWHILE"); return ENDWHILE;}



";"              {  print_return("QM"); return QM; }
```

```
"+="                {  print_return("ADD_ASSIGN"); return ADD_ASSIGN; }
"-="                {  print_return("SUB_ASSIGN"); return SUB_ASSIGN; }
"/="                {  print_return("DIV_ASSIGN"); return DIV_ASSIGN; }
"--"                {  print_return("DEC_OP"); return DEC_OP; }
"++"                {  print_return("INC_OP"); return INC_OP; }
"AND"               {  print_return("AND_OP"); return AND_OP; }
"OR"                {  print_return("OR_OP"); return OR_OP; }
"=="                {  print_return("EQ_OP"); return EQ_OP; }
">="                {  print_return("GE_OP"); return GE_OP; }
"<="                {  print_return("LE_OP"); return LE_OP; }
"!="                {  print_return("NE_OP"); return NE_OP; }
"{"                 {  print_return("L_BRACE"); return L_BRACE; }
"}"                 {  print_return("R_BRACE"); return R_BRACE; }
","                 {  print_return("COMMA"); return COMMA; }

"="                 {  print_return("ASSIGN"); return ASSIGN; }
"("                 {  print_return("L_PAR"); return L_PAR; }
")"                 {  print_return("R_PAR"); return R_PAR;}
"["                 {  print_return("L_BRACK"); return L_BRACK; }
"]"                 {  print_return("R_BRACK"); return R_BRACK;}
"."                 {  print_return("DOT"); return DOT; }
"_"                 {  print_return("UNDERSCORE"); return UNDERSCORE; }
"-"                 {  print_return("MINUS"); return MINUS; }
"+"                 {  print_return("PLUS"); return PLUS; }
"*"                 {  print_return("MUL"); return MUL; }
":"                 {  print_return("COLON"); return COLON; }

"/"                 {  print_return("DIV"); return DIV; }
"<"                 {  print_return("LT"); return LT; }
">"                 {  print_return("GT"); return GT; }
[ ]                 ;
.                   { yyerror("Unkown character"); }

{identifier}        { print_return("ID"); strcpy(yylval.name, yytext);  return IDENTIFIE
R; }
{integer}           { print_return("INTEGER"); yylval.integer_val = atoi(yytext); return
 INTEGER; }
{float_number}      { print_return("FLOAT"); return FLOAT; }
{char}              { print_return("CHAR"); return CHAR; }
{string}            { print_return("STRING"); return STRING; }
{NEWLINE}           {line_no++; print_return("NEWLINE"); return NEWLINE;}
{WHITESPACE}        {}

%%
/* ------------------------------------------------------------------------------
----- C FUNCTIONS ------------------------------------------------------------------
---------------------------------- */
```

```
void print_return(char *token)
{
    printf("Token: %s\t\t Line: %d\t\t Text: %s\n", token, line_no, yytext);
}
```

Αρχείο Parser.y

```
%{

    #include <stdio.h>
    #include <stdlib.h>
    #include <math.h>
    #include "print_console.c"

    //pointer to input file of lexer
    extern FILE *yyin;
    //pointer to output file of lexer
    extern FILE *yyout;
    //line counter
    extern int line_no;
    //reads the input stream generates tokens
    extern int yylex();
    //temporary token save
    extern char* yytext;

    //Function Initilize
    int yylex();
    void yyerror(char *message);

%}

//struct for print_console
%union
{
        char name[500];
        int integer_val;
}

/* --------------------------------------- TOKENS -----------------------------------
-----*/

%token COMMENT
%token ML_COMMENT
%token BREAK
%token VARS
%token QM

%token NEWLINE

%token STARTMAIN
%token ENDMAIN

%token IF
%token THEN
```

```
%token ELSEIF
%token ELSE
%token ENDIF

%token FOR
%token TO
%token STEP
%token ENDFOR

%token SWITCH
%token CASE
%token ENDSWITCH

%token RETURN

%token PRINT

%token WHILE
%token ENDWHILE

%token INDENT

%token FUNCTION
%token END_FUNCTION

%token ADD_ASSIGN
%token SUB_ASSIGN
%token DIV_ASSIGN

%token DEC_OP
%token INC_OP
%token AND_OP
%token OR_OP
%token EQ_OP
%token GE_OP
%token LE_OP
%token NE_OP

%token L_BRACE
%token R_BRACE
%token COMMA
%token COLON

%token ASSIGN
%token L_PAR
%token R_PAR
%token L_BRACK
%token R_BRACK
```

```
%token DOT
%token UNDERSCORE
%token MINUS
%token PLUS
%token MUL
%token DIV


%token LT
%token GT
%token FLOAT
%token CHAR
%token STRING

%token <name> IDENTIFIER
%token <integer_val> INTEGER


%token PROGRAM

//type for access to $$
%type <integer_val> line int_op int_data
%type <name> calc_assignment

%%

program: PROGRAM IDENTIFIER NEWLINE | program line | program start_main ;

line:   if_stmt {;}
        | elseif_stmt {;}
        | else_stmt {;}
        | for_statement {;}
        | function {;}
        | function_call {;}
        | comments  {;}
        | variable {;}
        | print  {;}
        | break {;}
        | inspector_gen {;}
        | switch  {;}
        | while  {;}
        | dictionaries  {;}
        | NEWLINE {;}
        | dictionary_data  {;}
        | calc_assignment  {;}
        | start_main {;}
        ;

/*--------- BREAK -------------*/
```

```
break: BREAK QM NEWLINE ;



/*--------- DATA TYPES -------------*/

data_type: CHAR
        | INTEGER
        | FLOAT
        | IDENTIFIER
        | STRING
        ;

/*--------- VARS -------------*/

variable: VARS  data_type inspector
        | VARS  data_type IDENTIFIER COMMA IDENTIFIER
        | VARS  variable_dictionary
        | variable variable_dictionary
        | variable QM
        | variable  variable_dictionary QM
        ;
variable_dictionary: data_type inspector
                     | data_type IDENTIFIER COMMA IDENTIFIER
                     | COMMA array
                     | array
                     | COMMA IDENTIFIER
                     | IDENTIFIER COMMA IDENTIFIER
                     | variable_dictionary COMMA IDENTIFIER
                     | variable_dictionary COMMA array
                     | line
                     ;

/*--------- RETURN -------------*/

return: RETURN INTEGER QM NEWLINE
        | RETURN IDENTIFIER QM NEWLINE
        | RETURN int_op QM NEWLINE
        ;

/*--------- FUNCTIONS --------------*/

function: FUNCTION IDENTIFIER L_PAR optional_parameters R_PAR NEWLINE line
          | function return end_function
          ;

end_function: END_FUNCTION NEWLINE;
```

```
function_call: IDENTIFIER L_PAR optional_parameters R_PAR
                | IDENTIFIER L_PAR data_type R_PAR
                | IDENTIFIER L_PAR data_type COMMA data_type R_PAR
                | IDENTIFIER L_PAR data_type COMMA data_type COMMA data_type R_PAR
                | function_call QM NEWLINE
                ;

/*------------ INSPECTORS -------------*/

inspector:IDENTIFIER operators data_type
        |data_type operators IDENTIFIER
        ;

inspector_gen: inspector
               | inspector operators
               | inspector operators inspector
               | inspector_gen QM
               ;


/*----------- IF & FOR STATEMENTS -------------*/

if_stmt: IF L_PAR inspector R_PAR THEN NEWLINE
        | if_stmt line
        | if_stmt end_if_stmt
        | if_stmt elseif_stmt
        | if_stmt elseif_stmt else_stmt
        ;
elseif_stmt: ELSEIF L_PAR inspector R_PAR  line ;
else_stmt: ELSE  line;
end_if_stmt: ENDIF NEWLINE;

for_statement: FOR IDENTIFIER COLON ASSIGN INTEGER TO INTEGER STEP INTEGER NEWLINE
               |for_statement line
               |for_statement end_for_statement
               ;
end_for_statement: ENDFOR NEWLINE;


/*---------- SWITCH / CASE STATEMENT -----------------*/

switch: SWITCH L_PAR LT IDENTIFIER GT R_PAR NEWLINE
        |SWITCH L_PAR LT IDENTIFIER GT R_PAR COMMENT
        |switch case
        |switch case end_switch
        ;

case:  CASE L_PAR LT INTEGER GT R_PAR NEWLINE line break;
end_switch: ENDSWITCH NEWLINE;
```

```
/*-------------- WHILE ---------------*/

while: WHILE L_PAR inspector_gen R_PAR NEWLINE
        |while line
        |while end_while
        ;
end_while: ENDWHILE NEWLINE;


/*-------------- ARRAY ---------------*/

array: IDENTIFIER L_BRACK INTEGER R_BRACK
        | IDENTIFIER L_BRACK IDENTIFIER R_BRACK
        ;
array_value: array ASSIGN INTEGER ;


/*-------------- OPERATORS & OPTIONAL PARAMETERS ---------------*/

operators:EQ_OP
      | GE_OP
      | LE_OP
      | NE_OP
      | DEC_OP
      | INC_OP
      | LT
      | GT
      | AND_OP
      | OR_OP
      ;


optional_parameters: IDENTIFIER
                     | data_type IDENTIFIER COMMA data_type IDENTIFIER
                     | IDENTIFIER IDENTIFIER COMMA optional_parameters
                     | IDENTIFIER IDENTIFIER
                     ;


/*-------------- COMMENTS ---------------*/

comments: COMMENT |  comments line | ml_comments;
ml_comments: ML_COMMENT ;


/*-------------- PRINT ---------------*/
```

```
print: PRINT L_PAR data_type R_PAR QM | PRINT L_PAR data_type print_name_var R_PAR Q
M | print NEWLINE;
print_name_var: L_BRACK COMMA IDENTIFIER R_BRACK
                | L_BRACK COMMA array R_BRACK
                ;

/*-------------- MAIN ---------------*/

start_main: STARTMAIN  NEWLINE
           | start_main line
           | start_main end_main
           ;
end_main: ENDMAIN NEWLINE;


/* ----------- DICTIONARIES ----------- */

dictionaries: IDENTIFIER ASSIGN L_BRACE dictionary_data R_BRACE
        | IDENTIFIER ASSIGN IDENTIFIER L_PAR L_BRACK L_PAR dictionary_data R_PAR R_B
RACK R_PAR
        | IDENTIFIER ASSIGN IDENTIFIER L_PAR dictionary_data optional_parameters dic
tionary_data R_PAR
        | IDENTIFIER ASSIGN function_call
        | array_value QM
        ;

dictionary_data: data_type COMMA data_type optional_parameters
                 | IDENTIFIER ASSIGN data_type QM
                 |/* empty */
                 ;



/* ----------- CALCULATE ------------ */

calc_assignment: IDENTIFIER ASSIGN int_op { Change($1, $3); }  ;

int_op: int_data { $$ = $1; }
        | int_op PLUS int_data { $$ = $1 + $3; }
        | int_op MINUS int_data { $$ = $1 - $3; }
        | int_op MUL int_data { $$ = $1 * $3; }
        | int_op DIV int_data { $$ = $1 / $3; }
        | int_op QM
        ;

int_data: INTEGER { $$ = $1; }
        | IDENTIFIER { $$ = Search($1) -> integer_val; }
        ;
```

```
%%

/* ------------------------------------------------ C FUNCTIONS --------------------
------------------------ */

void yyerror(char *message){
        printf("Error: \"%s\"\t in line %d. Token = %s\n", message, line_no, yytext)
;
        exit(1);
}

/* ------------------------------------------ MAIN FUNCTION -----------------------
--------------------- */

int main(int argc, char *argv[]){

        hashTable = (hash *) calloc(SIZE, sizeof(hash));

        int flag;

        yyin = fopen(argv[1],"r");
        //yyparse(): reads tokens, executes actions
        flag = yyparse();
        fclose(yyin);

        printf("Parsing finished succesfully!\n\n");
        printf(" __________________________\n");
        Print();
        printf(" __________________________\n");

        return flag;
}
```

Κάνοντας compile τα αρχεία parser.y και lexer.l *(-gcc lex.yy.c parser.tab.c)* στον lexer εισάγεται το αρχείο **parser.tab.h** μέσω της εντολής yylex η οποία αποτελεί τη συνάρτηση της λεξικής ανάλυσης και αναγνωρίζει τα tokens από το input stream και τα επιστρέφει στον parser. Καθώς ο Bison δεν δημιουργεί αυτήν την συνάρτηση αυτόματα πρέπει να τη γράψουμε ώστε να καλεστεί μέσω του yyparse. Το παραγώμενο αρχείο tab.h είναι το παρακάτω:

```
/* A Bison parser, made by GNU Bison 2.7.  */

/* Bison interface for Yacc-like parsers in C

      Copyright (C) 1984, 1989-1990, 2000-2012 Free Software Foundation, Inc.

   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.  */

/* As a special exception, you may create a larger work that contains
   part or all of the Bison parser skeleton and distribute that work
   under terms of your choice, so long as that work isn't itself a
   parser generator using the skeleton or a modified version thereof
   as a parser skeleton.  Alternatively, if you modify or redistribute
   the parser skeleton itself, you may (at your option) remove this
   special exception, which will cause the skeleton and the resulting
   Bison output files to be licensed under the GNU General Public
   License without this special exception.

   This special exception was added by the Free Software Foundation in
   version 2.2 of Bison.  */

#ifndef YY_YY_PARSER_TAB_H_INCLUDED
# define YY_YY_PARSER_TAB_H_INCLUDED
/* Enabling traces.  */
#ifndef YYDEBUG
# define YYDEBUG 0
#endif
#if YYDEBUG
extern int yydebug;
#endif
```

```
/* Tokens.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
   /* Put the tokens into the symbol table, so that GDB and other debuggers
      know about them.  */
   enum yytokentype {
     COMMENT = 258,
     ML_COMMENT = 259,
     BREAK = 260,
     VARS = 261,
     QM = 262,
     NEWLINE = 263,
     STARTMAIN = 264,
     ENDMAIN = 265,
     IF = 266,
     THEN = 267,
     ELSEIF = 268,
     ELSE = 269,
     ENDIF = 270,
     FOR = 271,
     TO = 272,
     STEP = 273,
     ENDFOR = 274,
     SWITCH = 275,
     CASE = 276,
     ENDSWITCH = 277,
     RETURN = 278,
     PRINT = 279,
     WHILE = 280,
     ENDWHILE = 281,
     INDENT = 282,
     FUNCTION = 283,
     END_FUNCTION = 284,
     ADD_ASSIGN = 285,
     SUB_ASSIGN = 286,
     DIV_ASSIGN = 287,
     DEC_OP = 288,
     INC_OP = 289,
     AND_OP = 290,
     OR_OP = 291,
     EQ_OP = 292,
     GE_OP = 293,
     LE_OP = 294,
     NE_OP = 295,
     L_BRACE = 296,
     R_BRACE = 297,
     COMMA = 298,
     COLON = 299,
     ASSIGN = 300,
     L_PAR = 301,
     R_PAR = 302,
     L_BRACK = 303,
     R_BRACK = 304,
     DOT = 305,
     UNDERSCORE = 306,
     MINUS = 307,
     PLUS = 308,
     MUL = 309,
     DIV = 310,
     LT = 311,
     GT = 312,
     FLOAT = 313,
     CHAR = 314,
     STRING = 315,
     IDENTIFIER = 316,
     INTEGER = 317,
     PROGRAM = 318
   };
#endif


#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
typedef union YYSTYPE
{
/* Line 2058 of yacc.c  */
#line 27 "parser.y"

   char name[500];
   int integer_val;


/* Line 2058 of yacc.c  */
#line 126 "parser.tab.h"
} YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define yystype YYSTYPE /* obsolescent; will be withdrawn */
# define YYSTYPE_IS_DECLARED 1
#endif

extern YYSTYPE yylval;

#ifdef YYPARSE_PARAM
#if defined __STDC__ || defined __cplusplus
int yyparse (void *YYPARSE_PARAM);
#else
int yyparse ();
```

```
#endif
#else /* ! YYPARSE_PARAM */
#if defined __STDC__ || defined __cplusplus
int yyparse (void);
#else
int yyparse ();
#endif
#endif /* ! YYPARSE_PARAM */

#endif /* !YY_YY_PARSER_TAB_H_INCLUDED  */
```

Αρχείο print_console.c

```c
#define SIZE 10

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

typedef struct node
{
    char name[100];
    int integer_val;
    struct node *next;
}var;

typedef struct hash
{
    var *head;
    int count;
}hash;

static hash *hashTable = NULL;


//memory allocation
var *newNode(char n[], int i)
{
    var *temp =  (var *)malloc(sizeof(var));

        strcpy(temp->name, n);
    temp->integer_val = i;
        temp->next = NULL;

        return temp;
};


//insert data into hash table
void Insert(char n[], int i)
{
    int hashIndex, h = 0;
    var *newnode = newNode(n, i);

    // hash function
    for (int c = 0; n[c] != '\0'; c++)
        h = (h + (unsigned char)n[c]);
    hashIndex = h % SIZE;

    if (!hashTable[hashIndex].head)
    {
        hashTable[hashIndex].head = newnode;
        hashTable[hashIndex].count = 1;
        return;
    }

    //adding new node to the list
    newnode->next = (hashTable[hashIndex].head);

    //update the head of the list and no of nodes in the current bucket
    hashTable[hashIndex].head = newnode;
    hashTable[hashIndex].count++;

    return;
};

//print the value of hashtable
void Print()
{
    var *myNode;
    int i;

    printf("| NAME\t\t | INTEGER |\n ");
    printf("__________________________\n");

    for (i = 0; i < SIZE; i++)
    {
            if (hashTable[i].count == 0)
                continue;

        myNode = hashTable[i].head;
            if (!myNode)
                continue;

            while (myNode != NULL)
        {
                printf("| %s\t\t | ", myNode->name);
            printf(" %d\t   |\n", myNode->integer_val);
                myNode = myNode->next;
            }
    }

    return;
};

//search value in hashtable and return the variable
var *Search(char n[])
{
        int hashIndex, h = 0, flag = 0;
        var *temp = NULL;

    for (int i = 0; n[i] != '\0'; i++)
        h = (h + (unsigned char)n[i]);

    hashIndex = h % SIZE;

    temp = hashTable[hashIndex].head;
    if (!temp) {
        printf("Search element not found in hash table\n");
        return temp;
    }

    while (temp != NULL) {
        if (strcmp(temp->name, n) == 0){
            flag = 1;
                break;
        }

            temp = temp->next;
    }

    if (!flag)
        printf("Search element not found in hash table\n");

    return temp;
}

//change the value
void Change(char n[], int i)
{
    int hashIndex, h = 0, flag = 0;
        var *temp;

        //hash function
    for (int i = 0; n[i] != '\0'; i++)
        h = (h + (unsigned char)n[i]);
    hashIndex = h % SIZE;

    temp = hashTable[hashIndex].head;
    if (!temp)
    {
        Insert(n, i);
        return;
    }

    while (temp != NULL)
    {
        if (strcmp(temp->name, n) == 0){
            temp->integer_val = i;
                flag = 1;
                break;
        }
        temp = temp->next;
    }

    if (!flag)
        printf("Search element not found in hash table\n");

        return;
};
```

Το αρχείο print_console.c ουσιαστικά υπάρχει για να διαβάζει τις νέες μεταβλητές που κάνει assign ο χρήστης και να αποθηκεύει το όνομά τους μαζί με την τιμή που αντιστοιχεί στην καθεμία σε ένα linked list που ονομάζεται hashtable.

Αρχικά δηλώνουμε το struck node, που θα κρατάει το όνομα της μεταβλητής στην μεταβλητή name[100] και το integer value της στην μεταβλητή integer_val, καθώς και θα κάνει link με το επόμενο struct τύπου node μέσω του pointer next. Επίσης, για να φτιάξουμε το hash table χρησιμοποιούμε ένα struct με όνομα hash, που δείχνει στο πρώτο στοιχείο μέσα στο table με τον pointer head και την μεταβλητή count για να «κρατάει» τον αριθμό κάθε στοιχείου στο hashtable

Έπειτα υπάρχει το function newNode() που προσθέτει ένα στοιχείο στην λίστα, συγκεκριμένα στην τελευταία θέση.

Τέλος, οι συναρτήσεις Insert(), Print(), Search(), Change() οι οποίες λειτουργούν ουσιαστικά όλο το hashTable, αφού αντίστοιχα εισάγουν ένα νέο στοιχείο, «εκτυπώνουν» τις μεταβλητές του προγράμματος μαζί με τις τελικές τιμές τους (αυτή η συνάρτηση καλείται στο τέλος κάθε προγράμματος που έτρεξε), αναζητούν ένα στοιχείο και αλλάζουν τα δεδομένα οποιουδήποτε στοιχείου μέσα στο hashtable.

Αρχείο εισαγωγής ψευδογλώσσας (input.cme)

```
PROGRAM input

/*----------------------------FLEX AND BISON PROJECT ----------------------------
-

--------------------------LEXICAL AND SYNTAX ANALYSIS ---------------------

------------------------------HERE'S OUR PROGRAM--------------------------------
----

---------------------------MULTIPLE LINE COMMENT TEST---------------------------
------------------*/

%function/if test

FUNCTION smaller(INTEGER x1, INTEGER x2)

    IF (x1<x2) THEN

        PRINT(""[,x1]);

    ELSEIF (x1<x2)

        PRINT(""[,x2]);

    ELSE

        PRINT("The two numbers are equal");

    ENDIF

    RETURN x1;

END_FUNCTION


%function/while test

FUNCTION doStuff(INTEGER var1, INTEGER var2)

    WHILE(var1<10 AND var2<20)

        IF(var1==var2) THEN

            BREAK;

        ENDIF

        var1 = 45;

        var2 >= 56;

    ENDWHILE
    RETURN var2;
END_FUNCTION


FUNCTION swissFiss(INTEGER day)
    SWITCH(<day>)
    CASE(<11>)
        PRINT("Monday");
        BREAK;
    CASE(<12>)
        PRINT("Tuesday");
        BREAK;
    CASE(<13>)
        PRINT("Wednesday");
        BREAK;
    CASE(<14>)
        PRINT("Thursday");
        BREAK;
    CASE(<15>)
        PRINT("Friday");
        BREAK;
    CASE(<16>)
        PRINT("Saturday");


        BREAK;
    CASE(<17>)
        PRINT("Sunday");
        BREAK;
    ENDSWITCH

    RETURN day;

END_FUNCTION




STARTMAIN

    VARS

        CHAR char1, char2;

        INTEGER varr, foo, foo1, foo2, foo3, foo4, foo5, pinakas[100], day;


    % calculations

    foo = 10;

    foo1 = 100;

    %foo2 = foo + foo1

    %foo3 = foo * foo2;

    foo4 = doStuff(foo3, foo1);


    plus = 50 + 60;

    mul = 10*20;

    div = 300/50;

    sub = 40-60;


    pinakas[10] = 10;

    pinakas[20] = 20;

    pinakas[30] = 30;

    pinakas[40] = 40;

    pinakas[50] = 50;
    pinakas[60] = 60;

    pinakas[70] = 70;

    pinakas[80] = 80;

    pinakas[90] = 90;

    pinakas[100] = 100;


    FOR varr:=10 TO 100 STEP 10

        PRINT(""[,pinakas[varr]]);

    ENDFOR


    %day = pinakas[50];


    %function_call

    swissFiss(day);

    %end of program.:)

ENDMAIN
```

Παράδειγμα Εκτέλεσης

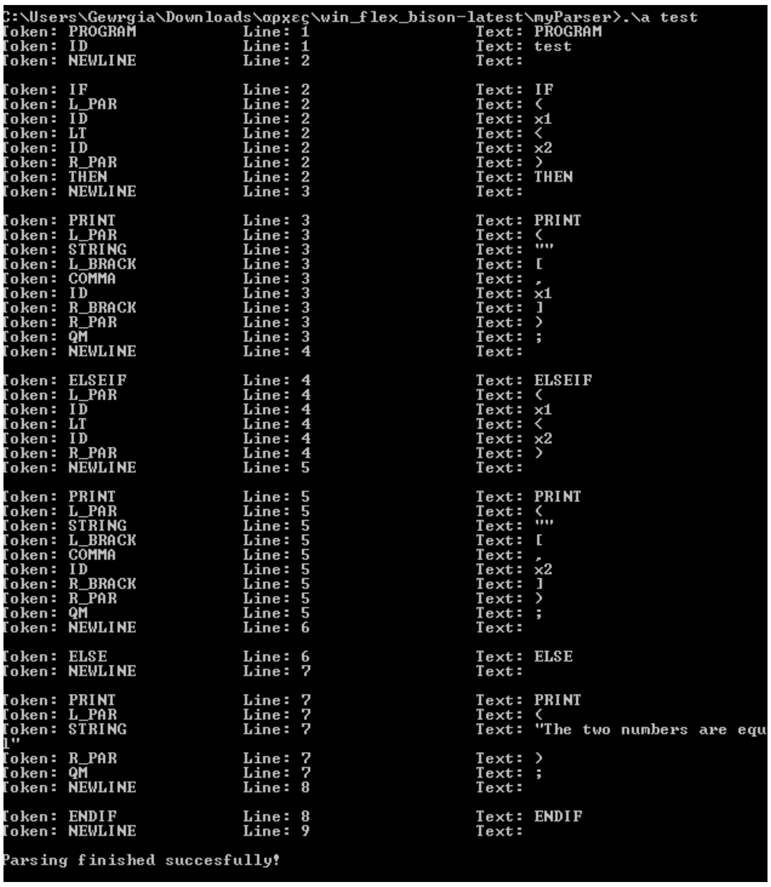
Πριν τρέξουμε το κύριο πρόγραμμα μας input.cme θα τρέξουμε μερικά παραδείγματα για κάθε ζητούμενο για να δείξουμε την λειτουργικότητα του προγράμματος σε ένα αρχείο test.

(Το πρόγραμμά μας ξεκινάει πάντα με την εντολή PROGRAM + όνομα δηλαδή σε αυτήν την περίπτωση PROGRAM test)

If function:

```
IF (x1<x2) THEN

    PRINT(""[,x1]);

  ELSEIF (x1<x2)

    PRINT(""[,x2]);

  ELSE

    PRINT("The two numbers are equal");

  ENDIF
```

Αποτέλεσμα:



```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM           Line: 1                 Text: PROGRAM
Token: ID                Line: 1                 Text: test
Token: NEWLINE           Line: 2                 Text:

Token: IF                Line: 2                 Text: IF
Token: L_PAR             Line: 2                 Text: (
Token: ID                Line: 2                 Text: x1
Token: LT                Line: 2                 Text: <
Token: ID                Line: 2                 Text: x2
Token: R_PAR             Line: 2                 Text: )
Token: THEN              Line: 2                 Text: THEN
Token: NEWLINE           Line: 3                 Text:

Token: PRINT             Line: 3                 Text: PRINT
Token: L_PAR             Line: 3                 Text: (
Token: STRING            Line: 3                 Text: ""
Token: L_BRACK           Line: 3                 Text: [
Token: COMMA             Line: 3                 Text: ,
Token: ID                Line: 3                 Text: x1
Token: R_BRACK           Line: 3                 Text: ]
Token: R_PAR             Line: 3                 Text: )
Token: QM                Line: 3                 Text: ;
Token: NEWLINE           Line: 4                 Text:

Token: ELSEIF            Line: 4                 Text: ELSEIF
Token: L_PAR             Line: 4                 Text: (
Token: ID                Line: 4                 Text: x1
Token: LT                Line: 4                 Text: <
Token: ID                Line: 4                 Text: x2
Token: R_PAR             Line: 4                 Text: )
Token: NEWLINE           Line: 5                 Text:

Token: PRINT             Line: 5                 Text: PRINT
Token: L_PAR             Line: 5                 Text: (
Token: STRING            Line: 5                 Text: ""
Token: L_BRACK           Line: 5                 Text: [
Token: COMMA             Line: 5                 Text: ,
Token: ID                Line: 5                 Text: x2
Token: R_BRACK           Line: 5                 Text: ]
Token: R_PAR             Line: 5                 Text: )
Token: QM                Line: 5                 Text: ;
Token: NEWLINE           Line: 6                 Text:

Token: ELSE              Line: 6                 Text: ELSE
Token: NEWLINE           Line: 7                 Text:

Token: PRINT             Line: 7                 Text: PRINT
Token: L_PAR             Line: 7                 Text: (
Token: STRING            Line: 7                 Text: "The two numbers are equ
l"
Token: R_PAR             Line: 7                 Text: )
Token: QM                Line: 7                 Text: ;
Token: NEWLINE           Line: 8                 Text:

Token: ENDIF             Line: 8                 Text: ENDIF
Token: NEWLINE           Line: 9                 Text:

Parsing finished succesfully!
```

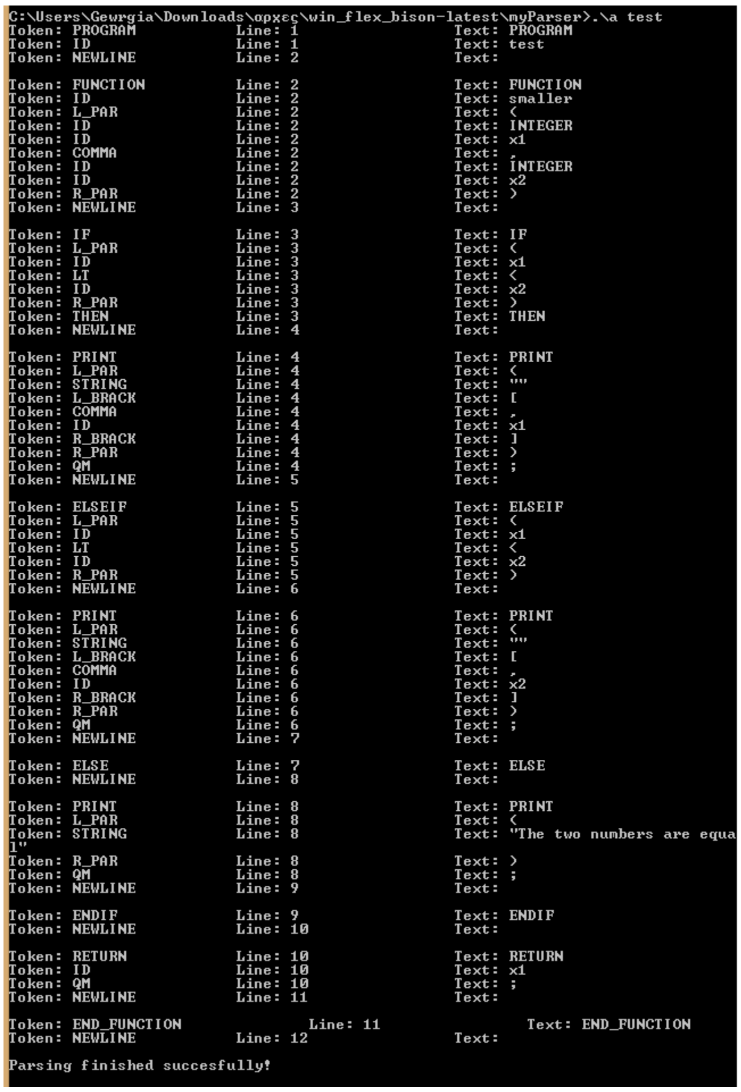
Όπως βλέπουμε τυπώνεται κατάλληλο μήνυμα ότι ο parser μας δουλεύει σωστά, “Parsing finished successfully!”

Κώδικας στη C στον οποίο οφείλεται:

```c
int main(int argc, char *argv[]){

        hashTable = (hash *) calloc(SIZE, sizeof(hash));

        int flag;

        yyin = fopen(argv[1],"r");
        //yyparse(): reads tokens, executes actions
        flag = yyparse();
        fclose(yyin);

        printf("Parsing finished succesfully!\n\n");
        printf(" __________________________\n");
        Print();
        printf(" __________________________\n");

        return flag;
}
```

If function μέσα σε function:

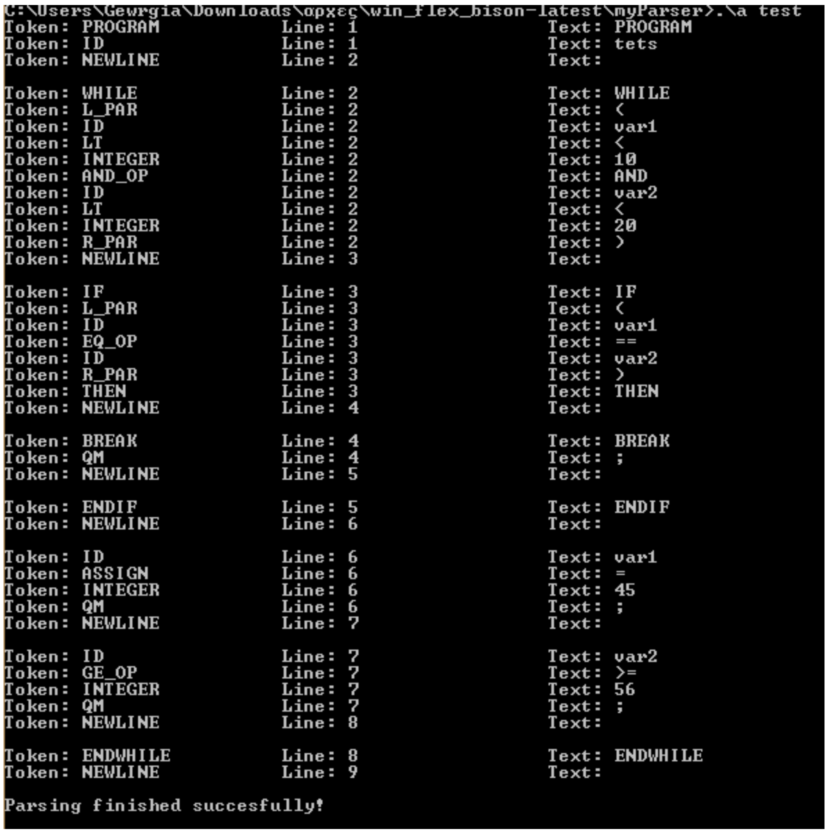
```
FUNCTION smaller(INTEGER x1, INTEGER x2)

  IF (x1<x2) THEN

    PRINT(""[,x1]);

  ELSEIF (x1<x2)

    PRINT(""[,x2]);

  ELSE

    PRINT("The two numbers are equal");

  ENDIF

  RETURN x1;

END_FUNCTION
```



```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM           Line: 1                 Text: PROGRAM
Token: ID                Line: 1                 Text: test
Token: NEWLINE           Line: 2                 Text:

Token: FUNCTION          Line: 2                 Text: FUNCTION
Token: ID                Line: 2                 Text: smaller
Token: L_PAR             Line: 2                 Text: (
Token: ID                Line: 2                 Text: INTEGER
Token: ID                Line: 2                 Text: x1
Token: COMMA             Line: 2                 Text: ,
Token: ID                Line: 2                 Text: INTEGER
Token: ID                Line: 2                 Text: x2
Token: R_PAR             Line: 2                 Text: )
Token: NEWLINE           Line: 3                 Text:

Token: IF                Line: 3                 Text: IF
Token: L_PAR             Line: 3                 Text: (
Token: ID                Line: 3                 Text: x1
Token: LT                Line: 3                 Text: <
Token: ID                Line: 3                 Text: x2
Token: R_PAR             Line: 3                 Text: )
Token: THEN              Line: 3                 Text: THEN
Token: NEWLINE           Line: 4                 Text:

Token: PRINT             Line: 4                 Text: PRINT
Token: L_PAR             Line: 4                 Text: (
Token: STRING            Line: 4                 Text: ""
Token: L_BRACK           Line: 4                 Text: [
Token: COMMA             Line: 4                 Text: ,
Token: ID                Line: 4                 Text: x1
Token: R_BRACK           Line: 4                 Text: ]
Token: R_PAR             Line: 4                 Text: )
Token: QM                Line: 4                 Text: ;
Token: NEWLINE           Line: 5                 Text:

Token: ELSEIF            Line: 5                 Text: ELSEIF
Token: L_PAR             Line: 5                 Text: (
Token: ID                Line: 5                 Text: x1
Token: LT                Line: 5                 Text: <
Token: ID                Line: 5                 Text: x2
Token: R_PAR             Line: 5                 Text: )
Token: NEWLINE           Line: 6                 Text:

Token: PRINT             Line: 6                 Text: PRINT
Token: L_PAR             Line: 6                 Text: (
Token: STRING            Line: 6                 Text: ""
Token: L_BRACK           Line: 6                 Text: [
Token: COMMA             Line: 6                 Text: ,
Token: ID                Line: 6                 Text: x2
Token: R_BRACK           Line: 6                 Text: ]
Token: R_PAR             Line: 6                 Text: )
Token: QM                Line: 6                 Text: ;
Token: NEWLINE           Line: 7                 Text:

Token: ELSE              Line: 7                 Text: ELSE
Token: NEWLINE           Line: 8                 Text:

Token: PRINT             Line: 8                 Text: PRINT
Token: L_PAR             Line: 8                 Text: (
Token: STRING            Line: 8                 Text: "The two numbers are equa
l"
Token: R_PAR             Line: 8                 Text: )
Token: QM                Line: 8                 Text: ;
Token: NEWLINE           Line: 9                 Text:

Token: ENDIF             Line: 9                 Text: ENDIF
Token: NEWLINE           Line: 10                Text:

Token: RETURN            Line: 10                Text: RETURN
Token: ID                Line: 10                Text: x1
Token: QM                Line: 10                Text: ;
Token: NEWLINE           Line: 11                Text:

Token: END_FUNCTION              Line: 11                Text: END_FUNCTION
Token: NEWLINE           Line: 12                Text:

Parsing finished succesfully!
```

While:

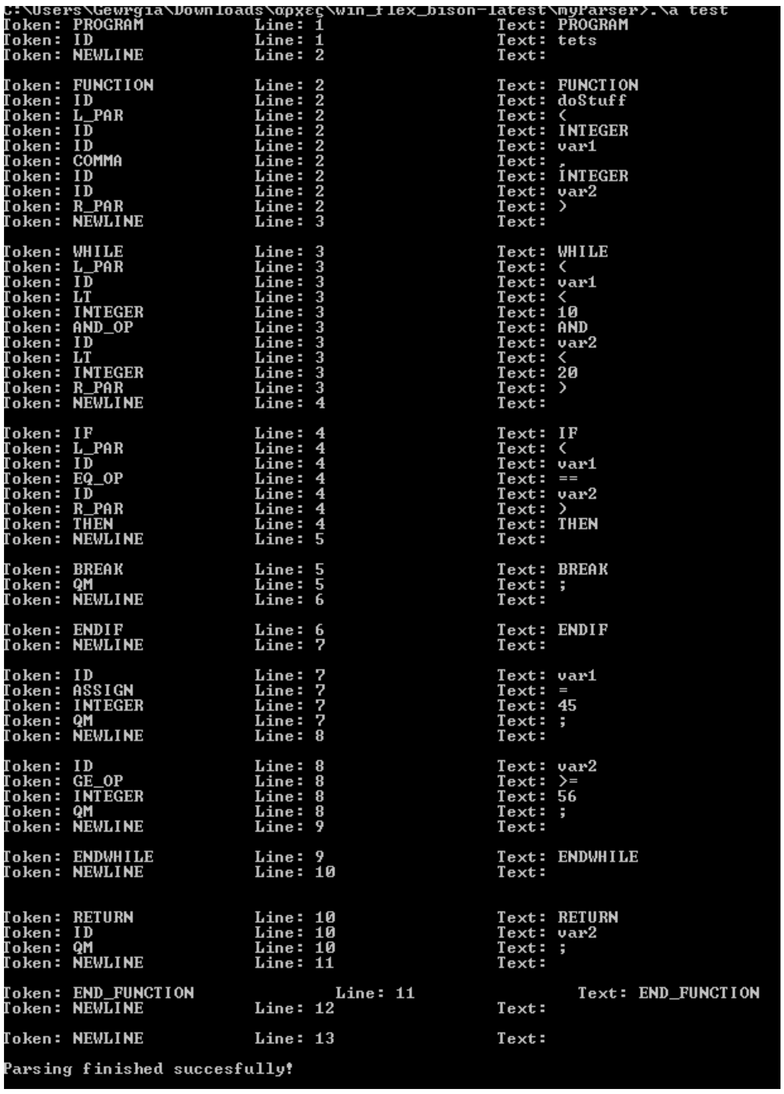
```
WHILE(var1<10 AND var2<20)

    IF(var1==var2) THEN

        BREAK;

    ENDIF

    var1 = 45;

    var2 >= 56;

  ENDWHILE
```



```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM              Line: 1                 Text: PROGRAM
Token: ID                   Line: 1                 Text: tets
Token: NEWLINE              Line: 2                 Text:

Token: WHILE                Line: 2                 Text: WHILE
Token: L_PAR                Line: 2                 Text: (
Token: ID                   Line: 2                 Text: var1
Token: LT                   Line: 2                 Text: <
Token: INTEGER              Line: 2                 Text: 10
Token: AND_OP               Line: 2                 Text: AND
Token: ID                   Line: 2                 Text: var2
Token: LT                   Line: 2                 Text: <
Token: INTEGER              Line: 2                 Text: 20
Token: R_PAR                Line: 2                 Text: )
Token: NEWLINE              Line: 3                 Text:

Token: IF                   Line: 3                 Text: IF
Token: L_PAR                Line: 3                 Text: (
Token: ID                   Line: 3                 Text: var1
Token: EQ_OP                Line: 3                 Text: ==
Token: ID                   Line: 3                 Text: var2
Token: R_PAR                Line: 3                 Text: )
Token: THEN                 Line: 3                 Text: THEN
Token: NEWLINE              Line: 4                 Text:

Token: BREAK                Line: 4                 Text: BREAK
Token: QM                   Line: 4                 Text: ;
Token: NEWLINE              Line: 5                 Text:

Token: ENDIF                Line: 5                 Text: ENDIF
Token: NEWLINE              Line: 6                 Text:

Token: ID                   Line: 6                 Text: var1
Token: ASSIGN               Line: 6                 Text: =
Token: INTEGER              Line: 6                 Text: 45
Token: QM                   Line: 6                 Text: ;
Token: NEWLINE              Line: 7                 Text:

Token: ID                   Line: 7                 Text: var2
Token: GE_OP                Line: 7                 Text: >=
Token: INTEGER              Line: 7                 Text: 56
Token: QM                   Line: 7                 Text: ;
Token: NEWLINE              Line: 8                 Text:

Token: ENDWHILE             Line: 8                 Text: ENDWHILE
Token: NEWLINE              Line: 9                 Text:

Parsing finished succesfully!
```

While in function:

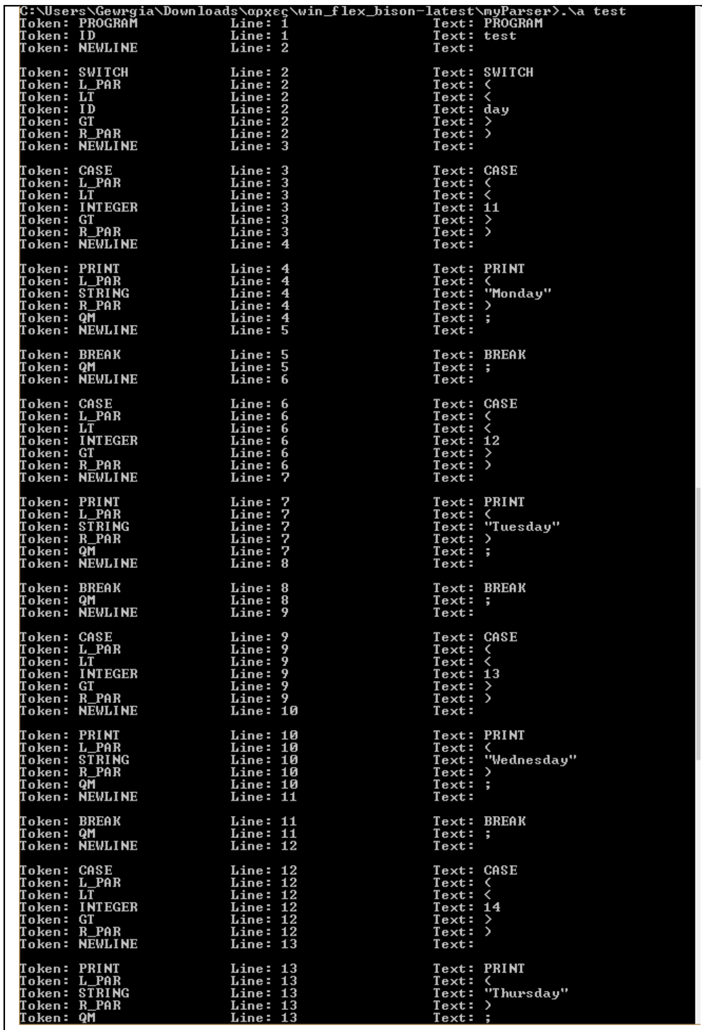
```
FUNCTION doStuff(INTEGER var1, INTEGER var2)

  WHILE(var1<10 AND var2<20)

    IF(var1==var2) THEN

      BREAK;

    ENDIF

    var1 = 45;

    var2 >= 56;

  ENDWHILE

  RETURN var2;

END_FUNCTION
```



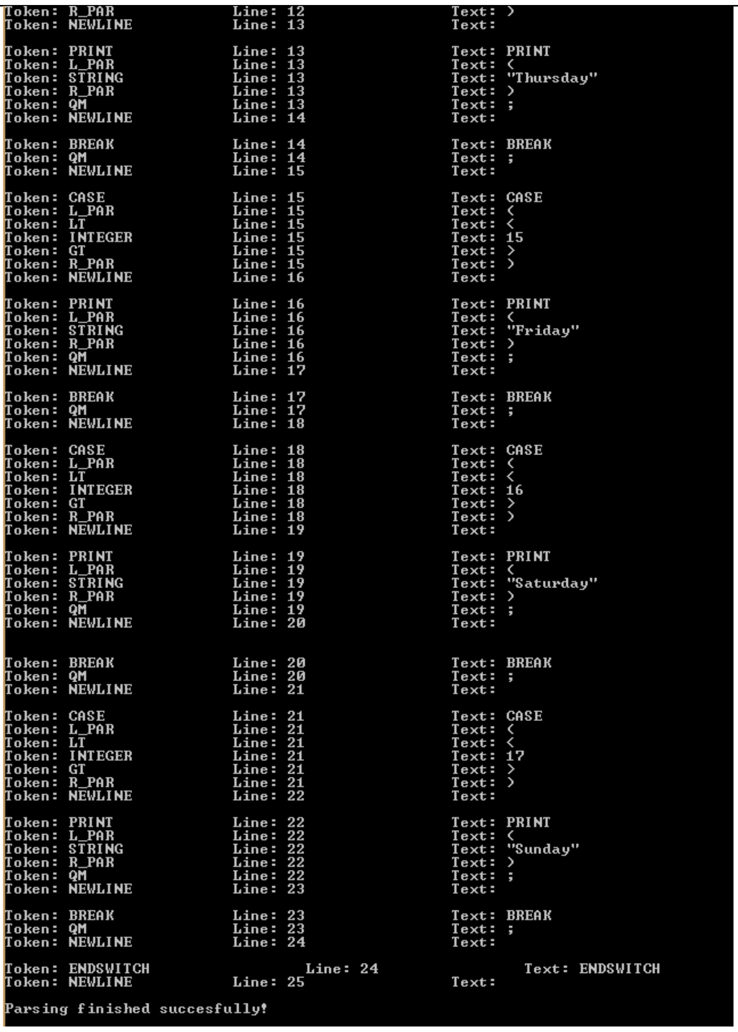
```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM           Line: 1                 Text: PROGRAM
Token: ID                Line: 1                 Text: tets
Token: NEWLINE           Line: 2                 Text:

Token: FUNCTION          Line: 2                 Text: FUNCTION
Token: ID                Line: 2                 Text: doStuff
Token: L_PAR             Line: 2                 Text: (
Token: ID                Line: 2                 Text: INTEGER
Token: ID                Line: 2                 Text: var1
Token: COMMA             Line: 2                 Text: ,
Token: ID                Line: 2                 Text: INTEGER
Token: ID                Line: 2                 Text: var2
Token: R_PAR             Line: 2                 Text: )
Token: NEWLINE           Line: 3                 Text:

Token: WHILE             Line: 3                 Text: WHILE
Token: L_PAR             Line: 3                 Text: (
Token: ID                Line: 3                 Text: var1
Token: LT                Line: 3                 Text: <
Token: INTEGER           Line: 3                 Text: 10
Token: AND_OP            Line: 3                 Text: AND
Token: ID                Line: 3                 Text: var2
Token: LT                Line: 3                 Text: <
Token: INTEGER           Line: 3                 Text: 20
Token: R_PAR             Line: 3                 Text: )
Token: NEWLINE           Line: 4                 Text:

Token: IF                Line: 4                 Text: IF
Token: L_PAR             Line: 4                 Text: (
Token: ID                Line: 4                 Text: var1
Token: EQ_OP             Line: 4                 Text: ==
Token: ID                Line: 4                 Text: var2
Token: R_PAR             Line: 4                 Text: )
Token: THEN              Line: 4                 Text: THEN
Token: NEWLINE           Line: 5                 Text:

Token: BREAK             Line: 5                 Text: BREAK
Token: QM                Line: 5                 Text: ;
Token: NEWLINE           Line: 6                 Text:

Token: ENDIF             Line: 6                 Text: ENDIF
Token: NEWLINE           Line: 7                 Text:

Token: ID                Line: 7                 Text: var1
Token: ASSIGN            Line: 7                 Text: =
Token: INTEGER           Line: 7                 Text: 45
Token: QM                Line: 7                 Text: ;
Token: NEWLINE           Line: 8                 Text:

Token: ID                Line: 8                 Text: var2
Token: GE_OP             Line: 8                 Text: >=
Token: INTEGER           Line: 8                 Text: 56
Token: QM                Line: 8                 Text: ;
Token: NEWLINE           Line: 9                 Text:

Token: ENDWHILE          Line: 9                 Text: ENDWHILE
Token: NEWLINE           Line: 10                Text:


Token: RETURN            Line: 10                Text: RETURN
Token: ID                Line: 10                Text: var2
Token: QM                Line: 10                Text: ;
Token: NEWLINE           Line: 11                Text:

Token: END_FUNCTION              Line: 11                Text: END_FUNCTION
Token: NEWLINE           Line: 12                Text:

Token: NEWLINE           Line: 13                Text:

Parsing finished succesfully!
```

Switch Case:

```
 SWITCH(<day>)

CASE(<11>)

  PRINT("Monday");

  BREAK;

CASE(<12>)

  PRINT("Tuesday");

  BREAK;

CASE(<13>)

  PRINT("Wednesday");

  BREAK;

CASE(<14>)

  PRINT("Thursday");

  BREAK;

CASE(<15>)

  PRINT("Friday");

  BREAK;

CASE(<16>)

  PRINT("Saturday");

  BREAK;

CASE(<17>)

  PRINT("Sunday");

  BREAK;

ENDSWITCH
```



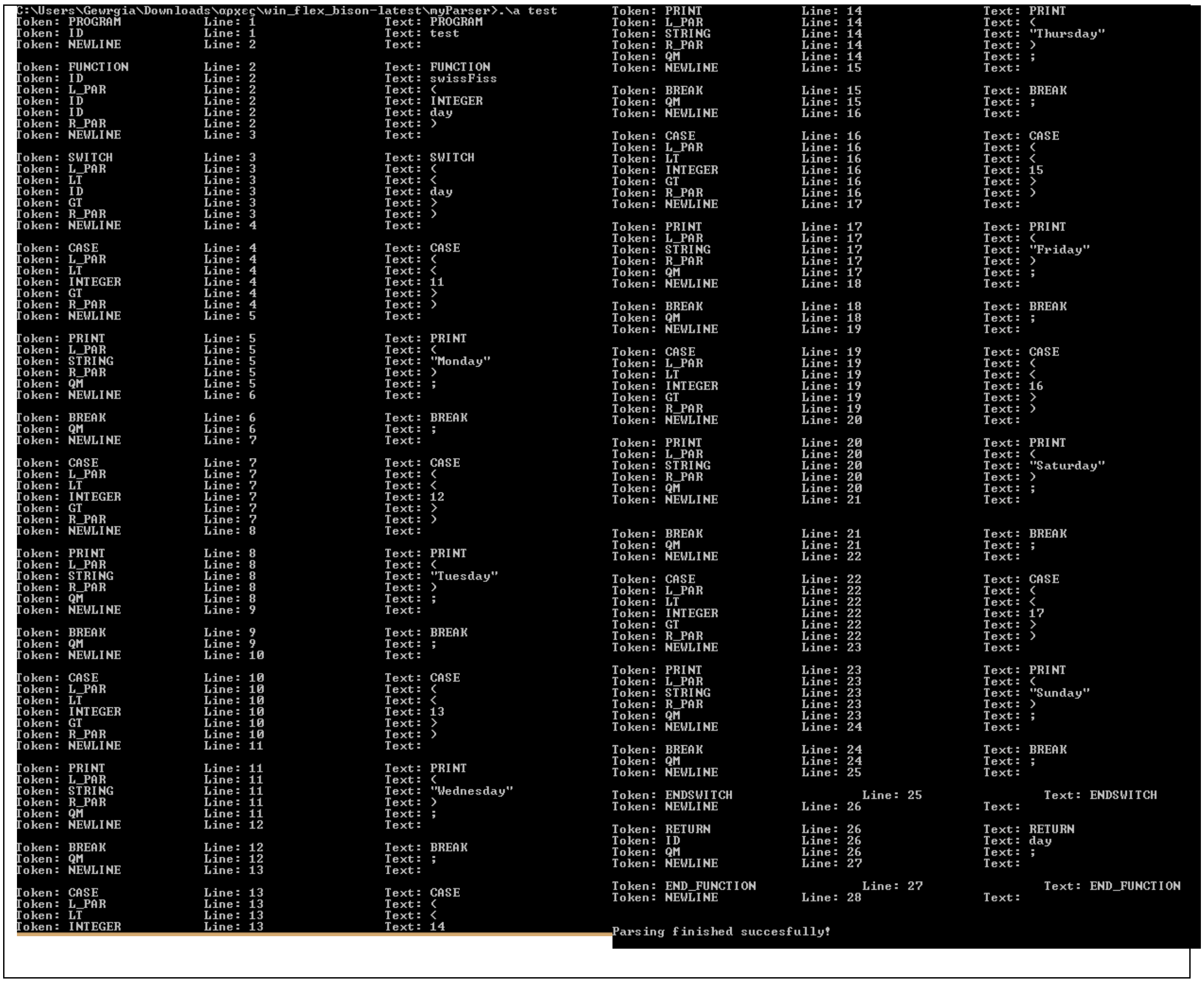
```
C:\Users\Gewrgia\Downloads\opxec\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM           Line: 1                 Text: PROGRAM
Token: ID                Line: 1                 Text: test
Token: NEWLINE           Line: 2                 Text:

Token: SWITCH            Line: 2                 Text: SWITCH
Token: L_PAR             Line: 2                 Text: (
Token: LT                Line: 2                 Text: <
Token: ID                Line: 2                 Text: day
Token: GT                Line: 2                 Text: >
Token: R_PAR             Line: 2                 Text: )
Token: NEWLINE           Line: 3                 Text:

Token: CASE              Line: 3                 Text: CASE
Token: L_PAR             Line: 3                 Text: (
Token: LT                Line: 3                 Text: <
Token: INTEGER           Line: 3                 Text: 11
Token: GT                Line: 3                 Text: >
Token: R_PAR             Line: 3                 Text: )
Token: NEWLINE           Line: 4                 Text:

Token: PRINT             Line: 4                 Text: PRINT
Token: L_PAR             Line: 4                 Text: (
Token: STRING            Line: 4                 Text: "Monday"
Token: R_PAR             Line: 4                 Text: )
Token: QM                Line: 4                 Text: ;
Token: NEWLINE           Line: 5                 Text:

Token: BREAK             Line: 5                 Text: BREAK
Token: QM                Line: 5                 Text: ;
Token: NEWLINE           Line: 6                 Text:

Token: CASE              Line: 6                 Text: CASE
Token: L_PAR             Line: 6                 Text: (
Token: LT                Line: 6                 Text: <
Token: INTEGER           Line: 6                 Text: 12
Token: GT                Line: 6                 Text: >
Token: R_PAR             Line: 6                 Text: )
Token: NEWLINE           Line: 7                 Text:

Token: PRINT             Line: 7                 Text: PRINT
Token: L_PAR             Line: 7                 Text: (
Token: STRING            Line: 7                 Text: "Tuesday"
Token: R_PAR             Line: 7                 Text: )
Token: QM                Line: 7                 Text: ;
Token: NEWLINE           Line: 8                 Text:

Token: BREAK             Line: 8                 Text: BREAK
Token: QM                Line: 8                 Text: ;
Token: NEWLINE           Line: 9                 Text:

Token: CASE              Line: 9                 Text: CASE
Token: L_PAR             Line: 9                 Text: (
Token: LT                Line: 9                 Text: <
Token: INTEGER           Line: 9                 Text: 13
Token: GT                Line: 9                 Text: >
Token: R_PAR             Line: 9                 Text: )
Token: NEWLINE           Line: 10                Text:

Token: PRINT             Line: 10                Text: PRINT
Token: L_PAR             Line: 10                Text: (
Token: STRING            Line: 10                Text: "Wednesday"
Token: R_PAR             Line: 10                Text: )
Token: QM                Line: 10                Text: ;
Token: NEWLINE           Line: 11                Text:

Token: BREAK             Line: 11                Text: BREAK
Token: QM                Line: 11                Text: ;
Token: NEWLINE           Line: 12                Text:

Token: CASE              Line: 12                Text: CASE
Token: L_PAR             Line: 12                Text: (
Token: LT                Line: 12                Text: <
Token: INTEGER           Line: 12                Text: 14
Token: GT                Line: 12                Text: >
Token: R_PAR             Line: 12                Text: )
Token: NEWLINE           Line: 13                Text:

Token: PRINT             Line: 13                Text: PRINT
Token: L_PAR             Line: 13                Text: (
Token: STRING            Line: 13                Text: "Thursday"
Token: R_PAR             Line: 13                Text: )
Token: QM                Line: 13                Text: ;
```



```
Token: R_PAR             Line: 12                Text: )
Token: NEWLINE           Line: 13                Text:

Token: PRINT             Line: 13                Text: PRINT
Token: L_PAR             Line: 13                Text: (
Token: STRING            Line: 13                Text: "Thursday"
Token: R_PAR             Line: 13                Text: )
Token: QM                Line: 13                Text: ;
Token: NEWLINE           Line: 14                Text:

Token: BREAK             Line: 14                Text: BREAK
Token: QM                Line: 14                Text: ;
Token: NEWLINE           Line: 15                Text:

Token: CASE              Line: 15                Text: CASE
Token: L_PAR             Line: 15                Text: (
Token: LT                Line: 15                Text: <
Token: INTEGER           Line: 15                Text: 15
Token: GT                Line: 15                Text: >
Token: R_PAR             Line: 15                Text: )
Token: NEWLINE           Line: 16                Text:

Token: PRINT             Line: 16                Text: PRINT
Token: L_PAR             Line: 16                Text: (
Token: STRING            Line: 16                Text: "Friday"
Token: R_PAR             Line: 16                Text: )
Token: QM                Line: 16                Text: ;
Token: NEWLINE           Line: 17                Text:

Token: BREAK             Line: 17                Text: BREAK
Token: QM                Line: 17                Text: ;
Token: NEWLINE           Line: 18                Text:

Token: CASE              Line: 18                Text: CASE
Token: L_PAR             Line: 18                Text: (
Token: LT                Line: 18                Text: <
Token: INTEGER           Line: 18                Text: 16
Token: GT                Line: 18                Text: >
Token: R_PAR             Line: 18                Text: )
Token: NEWLINE           Line: 19                Text:

Token: PRINT             Line: 19                Text: PRINT
Token: L_PAR             Line: 19                Text: (
Token: STRING            Line: 19                Text: "Saturday"
Token: R_PAR             Line: 19                Text: )
Token: QM                Line: 19                Text: ;
Token: NEWLINE           Line: 20                Text:


Token: BREAK             Line: 20                Text: BREAK
Token: QM                Line: 20                Text: ;
Token: NEWLINE           Line: 21                Text:

Token: CASE              Line: 21                Text: CASE
Token: L_PAR             Line: 21                Text: (
Token: LT                Line: 21                Text: <
Token: INTEGER           Line: 21                Text: 17
Token: GT                Line: 21                Text: >
Token: R_PAR             Line: 21                Text: )
Token: NEWLINE           Line: 22                Text:

Token: PRINT             Line: 22                Text: PRINT
Token: L_PAR             Line: 22                Text: (
Token: STRING            Line: 22                Text: "Sunday"
Token: R_PAR             Line: 22                Text: )
Token: QM                Line: 22                Text: ;
Token: NEWLINE           Line: 23                Text:

Token: BREAK             Line: 23                Text: BREAK
Token: QM                Line: 23                Text: ;
Token: NEWLINE           Line: 24                Text:

Token: ENDSWITCH                  Line: 24                Text: ENDSWITCH
Token: NEWLINE           Line: 25                Text:

Parsing finished succesfully!
```

Switch Case in Function:

```
FUNCTION swissFiss(INTEGER day)

  SWITCH(<day>)

  CASE(<11>)

    PRINT("Monday");

    BREAK;

  CASE(<12>)

    PRINT("Tuesday");

    BREAK;

  CASE(<13>)

    PRINT("Wednesday");

    BREAK;

  CASE(<14>)

    PRINT("Thursday");

    BREAK;

  CASE(<15>)

    PRINT("Friday");

    BREAK;

  CASE(<16>)

    PRINT("Saturday");


    BREAK;

  CASE(<17>)

    PRINT("Sunday");

    BREAK;

  ENDSWITCH

  RETURN day;

END_FUNCTION
```



```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM          Line: 1                 Text: PROGRAM
Token: ID               Line: 1                 Text: test
Token: NEWLINE          Line: 2                 Text:

Token: FUNCTION         Line: 2                 Text: FUNCTION
Token: ID               Line: 2                 Text: swissFiss
Token: L_PAR            Line: 2                 Text: (
Token: ID               Line: 2                 Text: INTEGER
Token: ID               Line: 2                 Text: day
Token: R_PAR            Line: 2                 Text: )
Token: NEWLINE          Line: 3                 Text:

Token: SWITCH           Line: 3                 Text: SWITCH
Token: L_PAR            Line: 3                 Text: (
Token: LT               Line: 3                 Text: <
Token: ID               Line: 3                 Text: day
Token: GT               Line: 3                 Text: >
Token: R_PAR            Line: 3                 Text: )
Token: NEWLINE          Line: 4                 Text:

Token: CASE             Line: 4                 Text: CASE
Token: L_PAR            Line: 4                 Text: (
Token: LT               Line: 4                 Text: <
Token: INTEGER          Line: 4                 Text: 11
Token: GT               Line: 4                 Text: >
Token: R_PAR            Line: 4                 Text: )
Token: NEWLINE          Line: 5                 Text:

Token: PRINT            Line: 5                 Text: PRINT
Token: L_PAR            Line: 5                 Text: (
Token: STRING           Line: 5                 Text: "Monday"
Token: R_PAR            Line: 5                 Text: )
Token: QM               Line: 5                 Text: ;
Token: NEWLINE          Line: 6                 Text:

Token: BREAK            Line: 6                 Text: BREAK
Token: QM               Line: 6                 Text: ;
Token: NEWLINE          Line: 7                 Text:

Token: CASE             Line: 7                 Text: CASE
Token: L_PAR            Line: 7                 Text: (
Token: LT               Line: 7                 Text: <
Token: INTEGER          Line: 7                 Text: 12
Token: GT               Line: 7                 Text: >
Token: R_PAR            Line: 7                 Text: )
Token: NEWLINE          Line: 8                 Text:

Token: PRINT            Line: 8                 Text: PRINT
Token: L_PAR            Line: 8                 Text: (
Token: STRING           Line: 8                 Text: "Tuesday"
Token: R_PAR            Line: 8                 Text: )
Token: QM               Line: 8                 Text: ;
Token: NEWLINE          Line: 9                 Text:

Token: BREAK            Line: 9                 Text: BREAK
Token: QM               Line: 9                 Text: ;
Token: NEWLINE          Line: 10                Text:

Token: CASE             Line: 10                Text: CASE
Token: L_PAR            Line: 10                Text: (
Token: LT               Line: 10                Text: <
Token: INTEGER          Line: 10                Text: 13
Token: GT               Line: 10                Text: >
Token: R_PAR            Line: 10                Text: )
Token: NEWLINE          Line: 11                Text:

Token: PRINT            Line: 11                Text: PRINT
Token: L_PAR            Line: 11                Text: (
Token: STRING           Line: 11                Text: "Wednesday"
Token: R_PAR            Line: 11                Text: )
Token: QM               Line: 11                Text: ;
Token: NEWLINE          Line: 12                Text:

Token: BREAK            Line: 12                Text: BREAK
Token: QM               Line: 12                Text: ;
Token: NEWLINE          Line: 13                Text:

Token: CASE             Line: 13                Text: CASE
Token: L_PAR            Line: 13                Text: (
Token: LT               Line: 13                Text: <
Token: INTEGER          Line: 13                Text: 14
```

```
Token: PRINT            Line: 14                Text: PRINT
Token: L_PAR            Line: 14                Text: (
Token: STRING           Line: 14                Text: "Thursday"
Token: R_PAR            Line: 14                Text: )
Token: QM               Line: 14                Text: ;
Token: NEWLINE          Line: 15                Text:

Token: BREAK            Line: 15                Text: BREAK
Token: QM               Line: 15                Text: ;
Token: NEWLINE          Line: 16                Text:

Token: CASE             Line: 16                Text: CASE
Token: L_PAR            Line: 16                Text: (
Token: LT               Line: 16                Text: <
Token: INTEGER          Line: 16                Text: 15
Token: GT               Line: 16                Text: >
Token: R_PAR            Line: 16                Text: )
Token: NEWLINE          Line: 17                Text:

Token: PRINT            Line: 17                Text: PRINT
Token: L_PAR            Line: 17                Text: (
Token: STRING           Line: 17                Text: "Friday"
Token: R_PAR            Line: 17                Text: )
Token: QM               Line: 17                Text: ;
Token: NEWLINE          Line: 18                Text:

Token: BREAK            Line: 18                Text: BREAK
Token: QM               Line: 18                Text: ;
Token: NEWLINE          Line: 19                Text:

Token: CASE             Line: 19                Text: CASE
Token: L_PAR            Line: 19                Text: (
Token: LT               Line: 19                Text: <
Token: INTEGER          Line: 19                Text: 16
Token: GT               Line: 19                Text: >
Token: R_PAR            Line: 19                Text: )
Token: NEWLINE          Line: 20                Text:

Token: PRINT            Line: 20                Text: PRINT
Token: L_PAR            Line: 20                Text: (
Token: STRING           Line: 20                Text: "Saturday"
Token: R_PAR            Line: 20                Text: )
Token: QM               Line: 20                Text: ;
Token: NEWLINE          Line: 21                Text:


Token: BREAK            Line: 21                Text: BREAK
Token: QM               Line: 21                Text: ;
Token: NEWLINE          Line: 22                Text:

Token: CASE             Line: 22                Text: CASE
Token: L_PAR            Line: 22                Text: (
Token: LT               Line: 22                Text: <
Token: INTEGER          Line: 22                Text: 17
Token: GT               Line: 22                Text: >
Token: R_PAR            Line: 22                Text: )
Token: NEWLINE          Line: 23                Text:

Token: PRINT            Line: 23                Text: PRINT
Token: L_PAR            Line: 23                Text: (
Token: STRING           Line: 23                Text: "Sunday"
Token: R_PAR            Line: 23                Text: )
Token: QM               Line: 23                Text: ;
Token: NEWLINE          Line: 24                Text:

Token: BREAK            Line: 24                Text: BREAK
Token: QM               Line: 24                Text: ;
Token: NEWLINE          Line: 25                Text:

Token: ENDSWITCH                Line: 25                Text: ENDSWITCH
Token: NEWLINE          Line: 26                Text:

Token: RETURN           Line: 26                Text: RETURN
Token: ID               Line: 26                Text: day
Token: QM               Line: 26                Text: ;
Token: NEWLINE          Line: 27                Text:

Token: END_FUNCTION             Line: 27                Text: END_FUNCTION
Token: NEWLINE          Line: 28                Text:

Parsing finished succesfully!
```

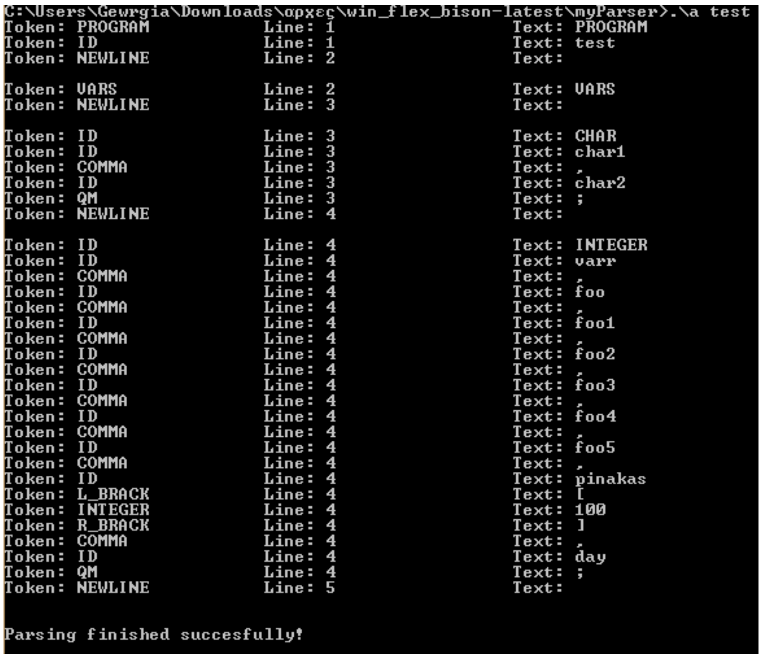
Επίσης, για τις συναρτήσεις βλέπουμε ότι υποχρεωτικά πριν το END_FUNCTION πρέπει να υπάρχει RETURN αλλιώς εμφανίζεται μήνυμα συντακτικού λάθους.

Vars: (Με δήλωση Chars και Integers)

VARS

CHAR char1, char2;

INTEGER varr, foo, foo1, foo2, foo3, foo4, foo5, pinakas[100], day;



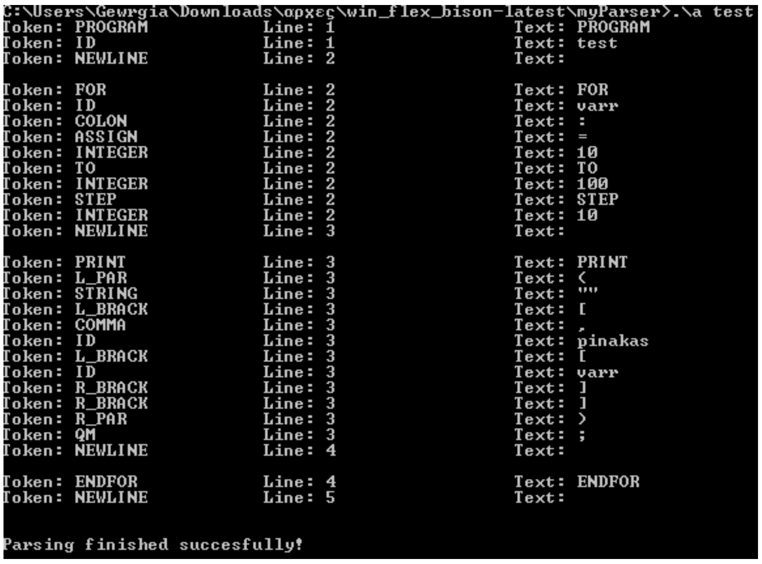
```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM             Line: 1                 Text: PROGRAM
Token: ID                  Line: 1                 Text: test
Token: NEWLINE             Line: 2                 Text:

Token: VARS                Line: 2                 Text: VARS
Token: NEWLINE             Line: 3                 Text:

Token: ID                  Line: 3                 Text: CHAR
Token: ID                  Line: 3                 Text: char1
Token: COMMA               Line: 3                 Text: ,
Token: ID                  Line: 3                 Text: char2
Token: QM                  Line: 3                 Text: ;
Token: NEWLINE             Line: 4                 Text:

Token: ID                  Line: 4                 Text: INTEGER
Token: ID                  Line: 4                 Text: varr
Token: COMMA               Line: 4                 Text: ,
Token: ID                  Line: 4                 Text: foo
Token: COMMA               Line: 4                 Text: ,
Token: ID                  Line: 4                 Text: foo1
Token: COMMA               Line: 4                 Text: ,
Token: ID                  Line: 4                 Text: foo2
Token: COMMA               Line: 4                 Text: ,
Token: ID                  Line: 4                 Text: foo3
Token: COMMA               Line: 4                 Text: ,
Token: ID                  Line: 4                 Text: foo4
Token: COMMA               Line: 4                 Text: ,
Token: ID                  Line: 4                 Text: foo5
Token: COMMA               Line: 4                 Text: ,
Token: ID                  Line: 4                 Text: pinakas
Token: L_BRACK             Line: 4                 Text: [
Token: INTEGER             Line: 4                 Text: 100
Token: R_BRACK             Line: 4                 Text: ]
Token: COMMA               Line: 4                 Text: ,
Token: ID                  Line: 4                 Text: day
Token: QM                  Line: 4                 Text: ;
Token: NEWLINE             Line: 5                 Text:


Parsing finished succesfully!
```

FOR: (Με δήλωση πίνακα)

FOR varr:=10 TO 100 STEP 10

PRINT(""[,pinakas[varr]]);

ENDFOR

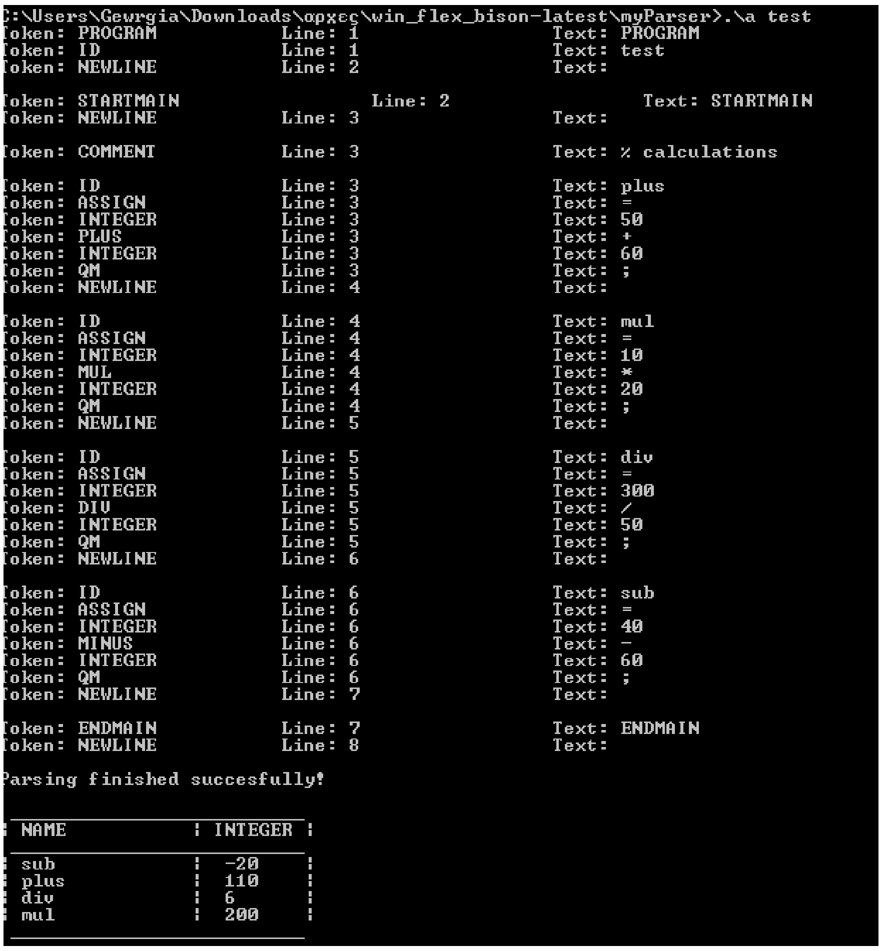


```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM             Line: 1                 Text: PROGRAM
Token: ID                  Line: 1                 Text: test
Token: NEWLINE             Line: 2                 Text:

Token: FOR                 Line: 2                 Text: FOR
Token: ID                  Line: 2                 Text: varr
Token: COLON               Line: 2                 Text: :
Token: ASSIGN              Line: 2                 Text: =
Token: INTEGER             Line: 2                 Text: 10
Token: TO                  Line: 2                 Text: TO
Token: INTEGER             Line: 2                 Text: 100
Token: STEP                Line: 2                 Text: STEP
Token: INTEGER             Line: 2                 Text: 10
Token: NEWLINE             Line: 3                 Text:

Token: PRINT               Line: 3                 Text: PRINT
Token: L_PAR               Line: 3                 Text: (
Token: STRING              Line: 3                 Text: ""
Token: L_BRACK             Line: 3                 Text: [
Token: COMMA               Line: 3                 Text: ,
Token: ID                  Line: 3                 Text: pinakas
Token: L_BRACK             Line: 3                 Text: [
Token: ID                  Line: 3                 Text: varr
Token: R_BRACK             Line: 3                 Text: ]
Token: R_BRACK             Line: 3                 Text: ]
Token: R_PAR               Line: 3                 Text: )
Token: QM                  Line: 3                 Text: ;
Token: NEWLINE             Line: 4                 Text:

Token: ENDFOR              Line: 4                 Text: ENDFOR
Token: NEWLINE             Line: 5                 Text:


Parsing finished succesfully!
```

Έπειτα, γράφουμε ένα παράδειγμα συνάρτησης main στην οποία εκτελούμε πράξεις. Τα αποτελέσματα αυτών των πράξεων αποθηκεύονται σε έναν πίνακα μαζί με τα ονόματα των μεταβλητών. Αυτό επιτεύχθηκε με το αρχείο print_console.c. Επίσης φαίνεται και η χρήση σχολίων *(%calculations)*.

Παράδειγμα **Main Function**:

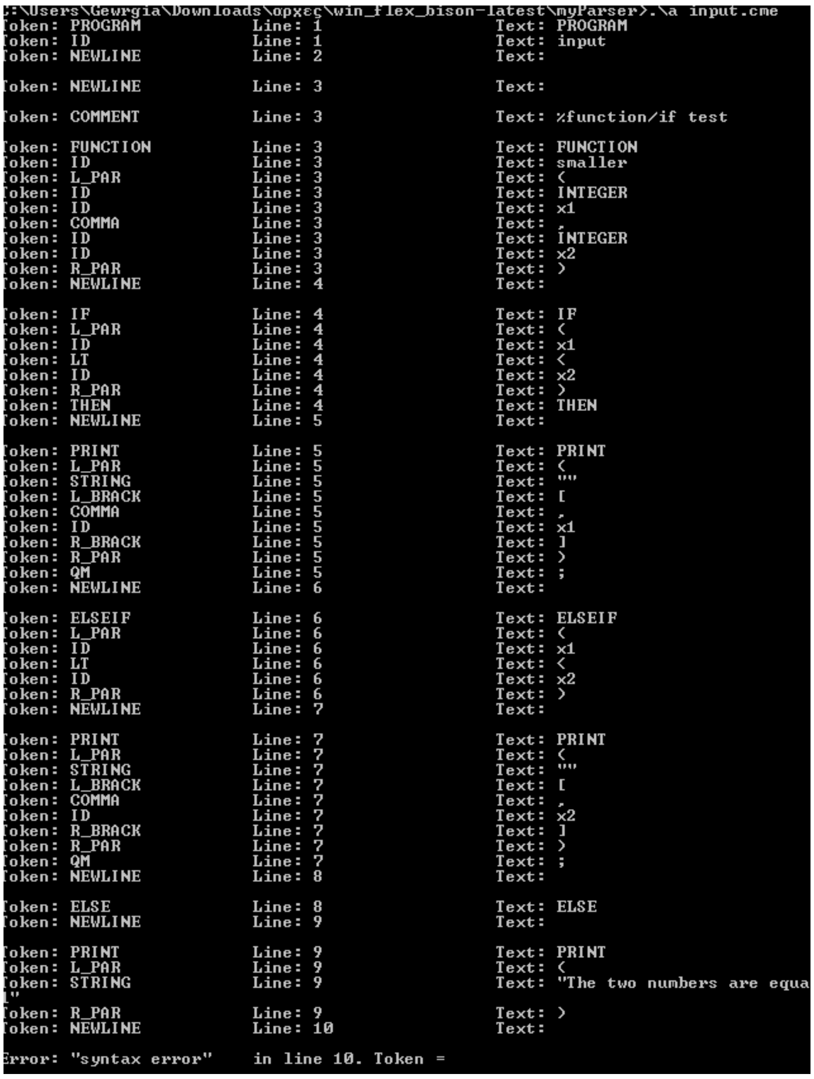
```
STARTMAIN

 % calculations

  plus = 50 + 60;

    mul = 10*20;

    div = 300/50;

    sub = 40-60;

ENDMAIN
```



```
:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
oken: PROGRAM           Line: 1               Text: PROGRAM
oken: ID                Line: 1               Text: test
oken: NEWLINE           Line: 2               Text:

oken: STARTMAIN                 Line: 2               Text: STARTMAIN
oken: NEWLINE           Line: 3               Text:

oken: COMMENT           Line: 3               Text: % calculations

oken: ID                Line: 3               Text: plus
oken: ASSIGN            Line: 3               Text: =
oken: INTEGER           Line: 3               Text: 50
oken: PLUS              Line: 3               Text: +
oken: INTEGER           Line: 3               Text: 60
oken: QM                Line: 3               Text: ;
oken: NEWLINE           Line: 4               Text:

oken: ID                Line: 4               Text: mul
oken: ASSIGN            Line: 4               Text: =
oken: INTEGER           Line: 4               Text: 10
oken: MUL               Line: 4               Text: *
oken: INTEGER           Line: 4               Text: 20
oken: QM                Line: 4               Text: ;
oken: NEWLINE           Line: 5               Text:

oken: ID                Line: 5               Text: div
oken: ASSIGN            Line: 5               Text: =
oken: INTEGER           Line: 5               Text: 300
oken: DIV               Line: 5               Text: /
oken: INTEGER           Line: 5               Text: 50
oken: QM                Line: 5               Text: ;
oken: NEWLINE           Line: 6               Text:

oken: ID                Line: 6               Text: sub
oken: ASSIGN            Line: 6               Text: =
oken: INTEGER           Line: 6               Text: 40
oken: MINUS             Line: 6               Text: -
oken: INTEGER           Line: 6               Text: 60
oken: QM                Line: 6               Text: ;
oken: NEWLINE           Line: 7               Text:

oken: ENDMAIN           Line: 7               Text: ENDMAIN
oken: NEWLINE           Line: 8               Text:

arsing finished succesfully!

 ________________________________________
: NAME              : INTEGER :
 ________________________________________
: sub               :  -20     :
: plus              :  110     :
: div               :  6       :
: mul               :  200     :
 ________________________________________
```

Έλεγχος λαθών

Τώρα, τρέχοντας ολόκληρο το πρόγραμμά μας εξετάζουμε αν αναγνωρίζει ορθά τα συντακτικά λάθη. Για παράδειγμα, αφαιρέσαμε σκόπημα το ερωτηματικό (;) από την εντολή print στην πρώτη function του προγράμματος και βλέπουμε ότι ο μεταγλωτισστής εντοπίζει σωστά που βρίσκεται το συντακτικό λάθος.



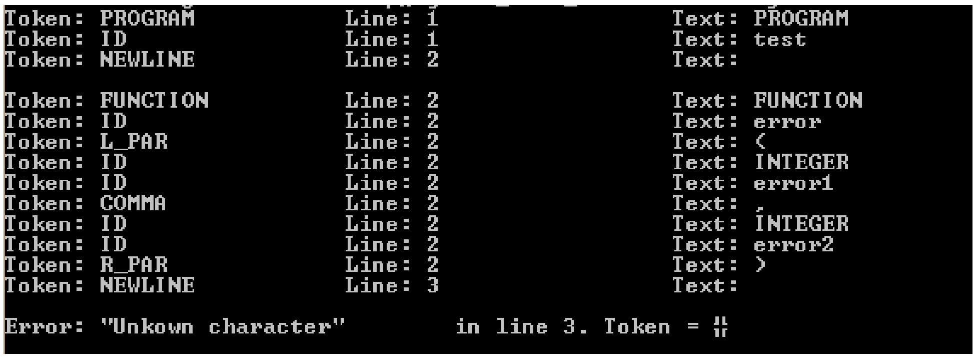
```
:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a input.cme
oken: PROGRAM            Line: 1                 Text: PROGRAM
oken: ID                 Line: 1                 Text: input
oken: NEWLINE            Line: 2                 Text:

oken: NEWLINE            Line: 3                 Text:

oken: COMMENT            Line: 3                 Text: %function/if test

oken: FUNCTION           Line: 3                 Text: FUNCTION
oken: ID                 Line: 3                 Text: smaller
oken: L_PAR              Line: 3                 Text: (
oken: ID                 Line: 3                 Text: INTEGER
oken: ID                 Line: 3                 Text: x1
oken: COMMA              Line: 3                 Text: ,
oken: ID                 Line: 3                 Text: INTEGER
oken: ID                 Line: 3                 Text: x2
oken: R_PAR              Line: 3                 Text: )
oken: NEWLINE            Line: 4                 Text:

oken: IF                 Line: 4                 Text: IF
oken: L_PAR              Line: 4                 Text: (
oken: ID                 Line: 4                 Text: x1
oken: LT                 Line: 4                 Text: <
oken: ID                 Line: 4                 Text: x2
oken: R_PAR              Line: 4                 Text: )
oken: THEN               Line: 4                 Text: THEN
oken: NEWLINE            Line: 5                 Text:

oken: PRINT              Line: 5                 Text: PRINT
oken: L_PAR              Line: 5                 Text: (
oken: STRING             Line: 5                 Text: ""
oken: L_BRACK            Line: 5                 Text: [
oken: COMMA              Line: 5                 Text: ,
oken: ID                 Line: 5                 Text: x1
oken: R_BRACK            Line: 5                 Text: ]
oken: R_PAR              Line: 5                 Text: )
oken: QM                 Line: 5                 Text: ;
oken: NEWLINE            Line: 6                 Text:

oken: ELSEIF             Line: 6                 Text: ELSEIF
oken: L_PAR              Line: 6                 Text: (
oken: ID                 Line: 6                 Text: x1
oken: LT                 Line: 6                 Text: <
oken: ID                 Line: 6                 Text: x2
oken: R_PAR              Line: 6                 Text: )
oken: NEWLINE            Line: 7                 Text:

oken: PRINT              Line: 7                 Text: PRINT
oken: L_PAR              Line: 7                 Text: (
oken: STRING             Line: 7                 Text: ""
oken: L_BRACK            Line: 7                 Text: [
oken: COMMA              Line: 7                 Text: ,
oken: ID                 Line: 7                 Text: x2
oken: R_BRACK            Line: 7                 Text: ]
oken: R_PAR              Line: 7                 Text: )
oken: QM                 Line: 7                 Text: ;
oken: NEWLINE            Line: 8                 Text:

oken: ELSE               Line: 8                 Text: ELSE
oken: NEWLINE            Line: 9                 Text:

oken: PRINT              Line: 9                 Text: PRINT
oken: L_PAR              Line: 9                 Text: (
oken: STRING             Line: 9                 Text: "The two numbers are equa
l"
oken: R_PAR              Line: 9                 Text: )
oken: NEWLINE            Line: 10                Text:

rror: "syntax error"     in line 10. Token =
```

Και εμφανίζεται κατάλληλο μήνυμα Syntax error.

Αυτό οφείλεται στην συνάρτηση void της C στο αρχείο του parser η οποία καλώντας την yyerror() εμφανίζει κατάλληλο μήνυμα ανάλογα με το λάθος.

```
void yyerror(char *message){
        printf("Error: \"%s\"\t in line %d. Token = %s\n", message, line_no, yytext)
;
        exit(1);
}
```

Στην παραπάνω περίπτωση είχαμε συντακτικό λάθος. Έστω ότι φτιάχνουμε μία συνάρτηση με περιεχόμενο τον ελληνικό χαρακτήρα 'ε'. Ο parser θα τυπώσει:



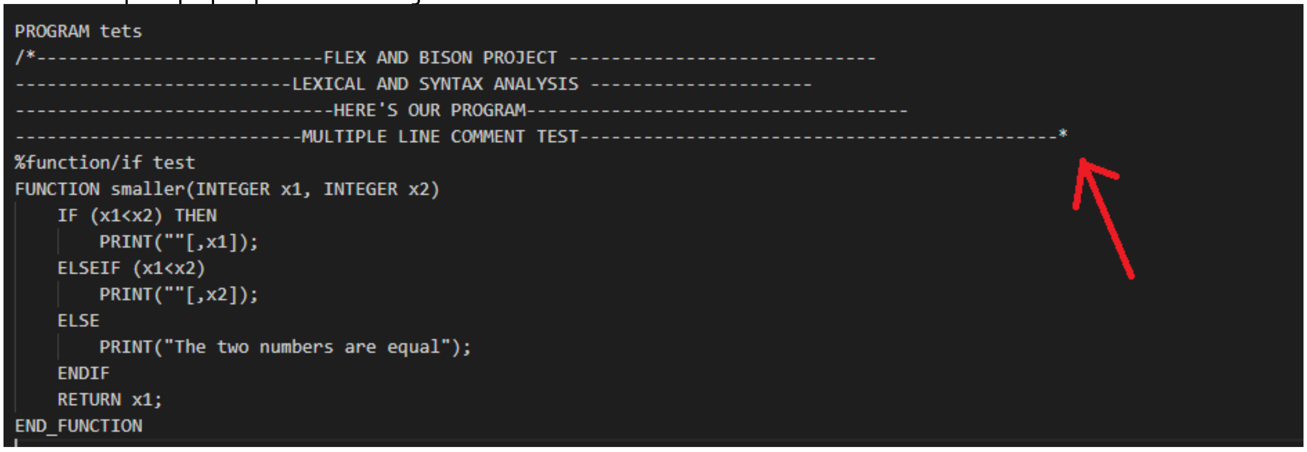
```
Token: PROGRAM          Line: 1                Text: PROGRAM
Token: ID               Line: 1                Text: test
Token: NEWLINE          Line: 2                Text:

Token: FUNCTION         Line: 2                Text: FUNCTION
Token: ID               Line: 2                Text: error
Token: L_PAR            Line: 2                Text: (
Token: ID               Line: 2                Text: INTEGER
Token: ID               Line: 2                Text: error1
Token: COMMA            Line: 2                Text: ,
Token: ID               Line: 2                Text: INTEGER
Token: ID               Line: 2                Text: error2
Token: R_PAR            Line: 2                Text: )
Token: NEWLINE          Line: 3                Text:

Error: "Unkown character"        in line 3. Token = ╬
```

Με μήνυμα λάθους "Unknown character" όπως έχει δηλωθεί στον lexer

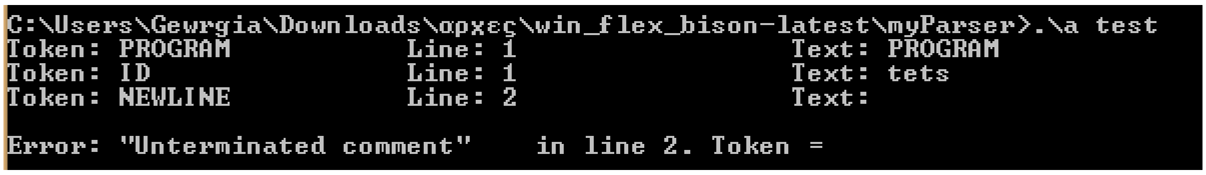
```
.                { yyerror("Unkown character"); }
```

στην περίπτωση εισαγωγής άγνωστου χαρακτήρα.

Στο project μας προσθέσαμε την δυνατότητα εισαγωγής σχολίων πολλαπλών γραμμών, χωρίς όμως να τυπώνονται ως tokens στο parsing. Κάνοντας όμως το σκόπιμο λάθος να αφαιρέσουμε το * ή το / από το τέλος του σχολίου εμφανίζεται το κατάλληλο μήνυμα λάθους:



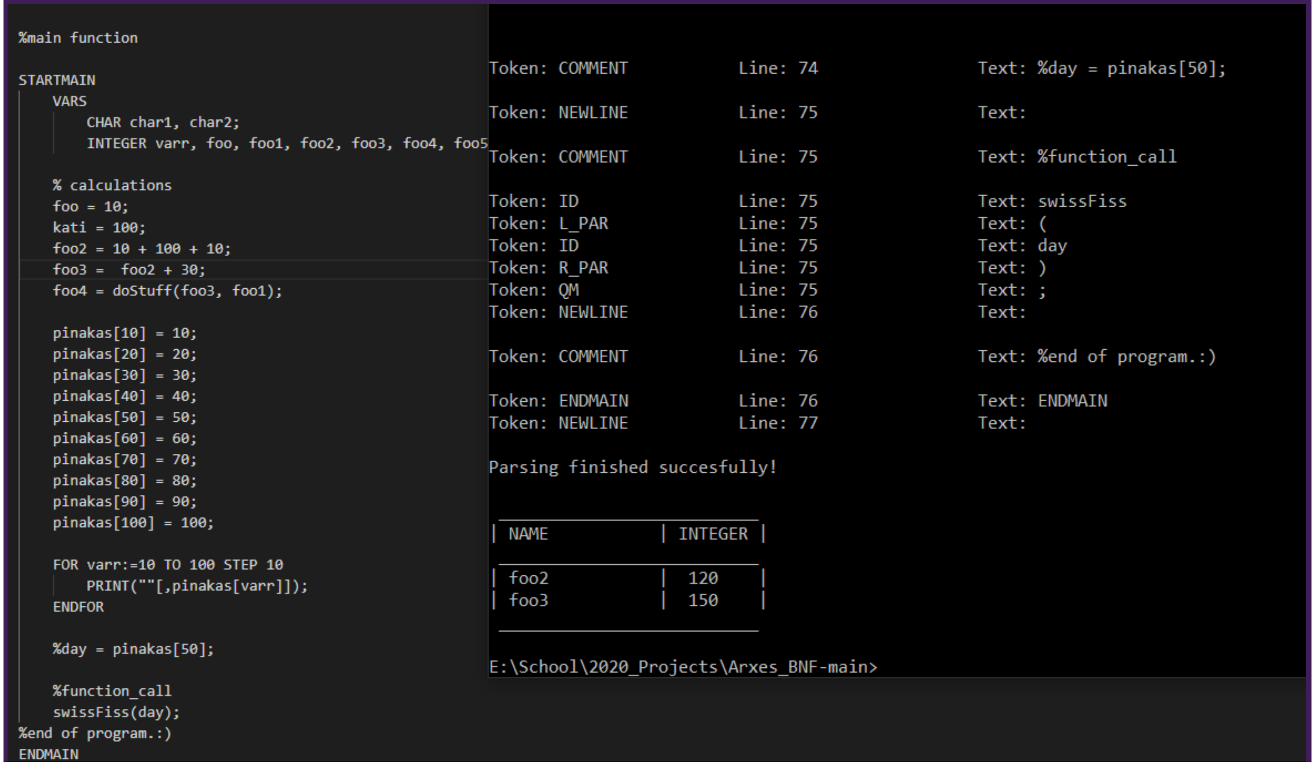
```
PROGRAM tets
/*--------------------------FLEX AND BISON PROJECT ----------------------------
-------------------------LEXICAL AND SYNTAX ANALYSIS --------------------
-----------------------------HERE'S OUR PROGRAM-----------------------------------
--------------------------MULTIPLE LINE COMMENT TEST--------------------------------------------*
%function/if test
FUNCTION smaller(INTEGER x1, INTEGER x2)
    IF (x1<x2) THEN
        PRINT(""[,x1]);
    ELSEIF (x1<x2)
        PRINT(""[,x2]);
    ELSE
        PRINT("The two numbers are equal");
    ENDIF
    RETURN x1;
END_FUNCTION
```



```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a test
Token: PROGRAM          Line: 1                Text: PROGRAM
Token: ID               Line: 1                Text: tets
Token: NEWLINE          Line: 2                Text:

Error: "Unterminated comment"    in line 2. Token =
```

Error: "Unterminated Comment".

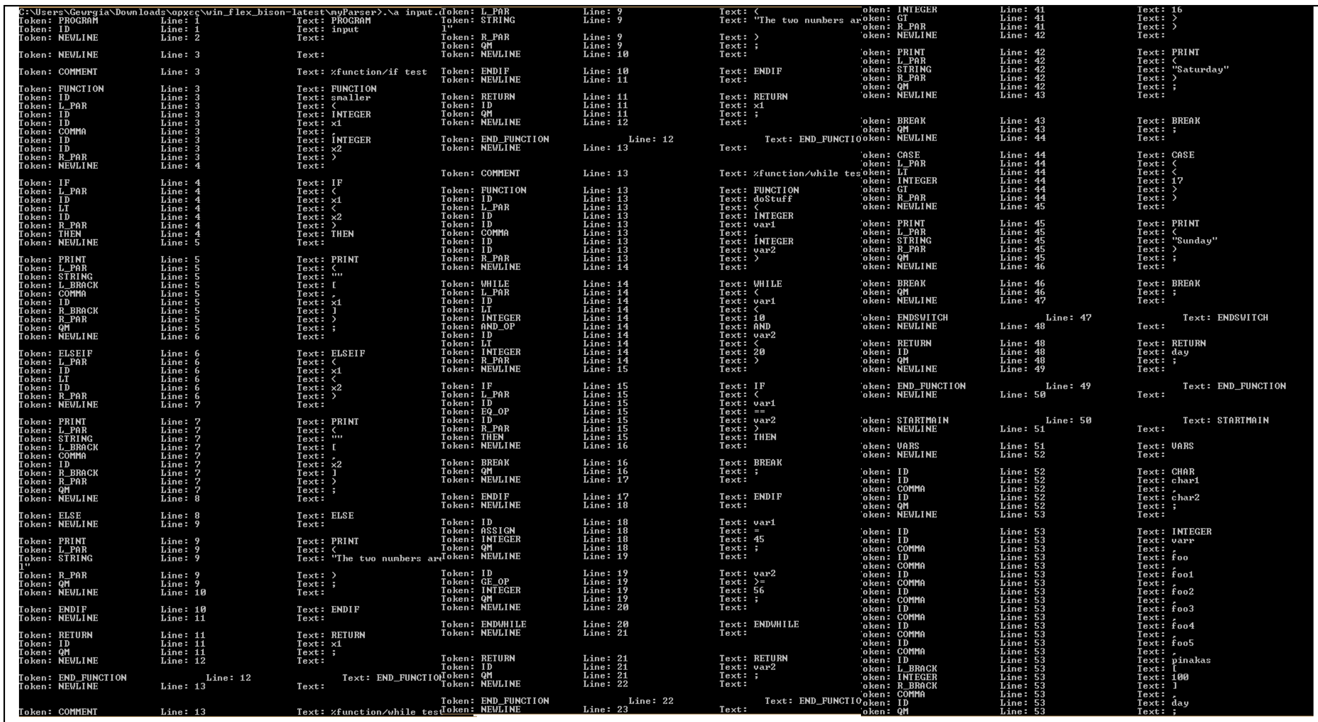
Αποτέλεσμα με πράξεις (αποθήκευση τιμής σε μεταβλητή):



```
%main function

STARTMAIN
    VARS
        CHAR char1, char2;
        INTEGER varr, foo, foo1, foo2, foo3, foo4, foo5

    % calculations
    foo = 10;
    kati = 100;
    foo2 = 10 + 100 + 10;
    foo3 =  foo2 + 30;
    foo4 = doStuff(foo3, foo1);

    pinakas[10] = 10;
    pinakas[20] = 20;
    pinakas[30] = 30;
    pinakas[40] = 40;
    pinakas[50] = 50;
    pinakas[60] = 60;
    pinakas[70] = 70;
    pinakas[80] = 80;
    pinakas[90] = 90;
    pinakas[100] = 100;

    FOR varr:=10 TO 100 STEP 10
        PRINT(""[,pinakas[varr]]);
    ENDFOR

    %day = pinakas[50];

    %function_call
    swissFiss(day);
%end of program.:)
ENDMAIN
```

```
Token: COMMENT          Line: 74                Text: %day = pinakas[50];

Token: NEWLINE          Line: 75                Text:

Token: COMMENT          Line: 75                Text: %function_call

Token: ID               Line: 75                Text: swissFiss
Token: L_PAR            Line: 75                Text: (
Token: ID               Line: 75                Text: day
Token: R_PAR            Line: 75                Text: )
Token: QM               Line: 75                Text: ;
Token: NEWLINE          Line: 76                Text:

Token: COMMENT          Line: 76                Text: %end of program.:)

Token: ENDMAIN          Line: 76                Text: ENDMAIN
Token: NEWLINE          Line: 77                Text:

Parsing finished succesfully!

 _________________________
| NAME           | INTEGER |
 _________________________
| foo2           |  120    |
| foo3           |  150    |
 _________________________

E:\School\2020_Projects\Arxes_BNF-main>
```

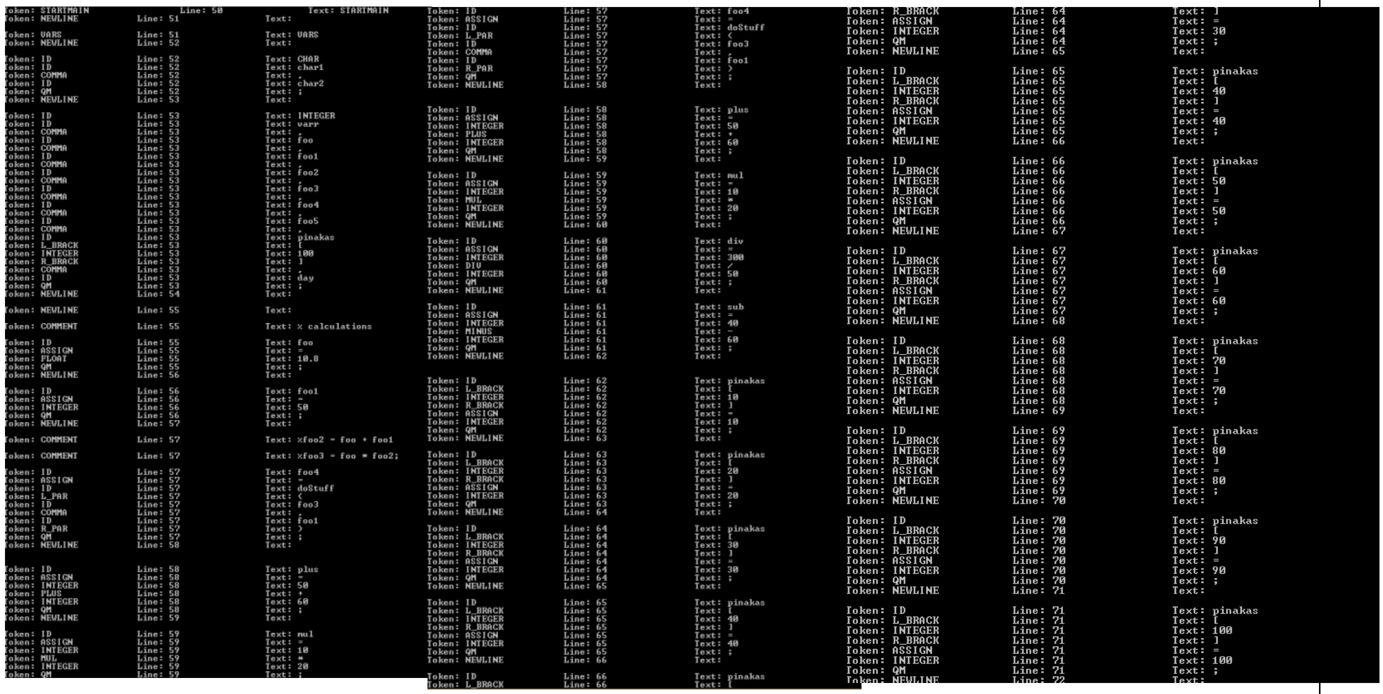
Αποτελέσματα του αρχείου input.cme:



```
C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>.\a input.
Token: PROGRAM          Line: 1          Text: PROGRAM
Token: ID               Line: 1          Text: input
Token: NEWLINE          Line: 2          Text:

Token: NEWLINE          Line: 3          Text:

Token: COMMENT          Line: 3          Text: %function/if test

Token: FUNCTION         Line: 3          Text: FUNCTION
Token: ID               Line: 3          Text: smaller
Token: L_PAR            Line: 3          Text: (
Token: ID               Line: 3          Text: INTEGER
Token: ID               Line: 3          Text: x1
Token: COMMA            Line: 3          Text: ,
Token: ID               Line: 3          Text: INTEGER
Token: ID               Line: 3          Text: x2
Token: R_PAR            Line: 3          Text: )
Token: NEWLINE          Line: 4          Text:

Token: IF               Line: 4          Text: IF
Token: L_PAR            Line: 4          Text: (
Token: ID               Line: 4          Text: x1
Token: LT               Line: 4          Text: <
Token: ID               Line: 4          Text: x2
Token: R_PAR            Line: 4          Text: )
Token: THEN             Line: 4          Text: THEN
Token: NEWLINE          Line: 5          Text:

Token: PRINT            Line: 5          Text: PRINT
Token: L_PAR            Line: 5          Text: (
Token: STRING           Line: 5          Text: ""
Token: L_BRACK          Line: 5          Text: [
Token: COMMA            Line: 5          Text: ,
Token: ID               Line: 5          Text: x1
Token: R_BRACK          Line: 5          Text: ]
Token: R_PAR            Line: 5          Text: )
Token: QM               Line: 5          Text: ;
Token: NEWLINE          Line: 6          Text:

Token: ELSEIF           Line: 6          Text: ELSEIF
Token: L_PAR            Line: 6          Text: (
Token: ID               Line: 6          Text: x1
Token: LT               Line: 6          Text: <
Token: ID               Line: 6          Text: x2
Token: R_PAR            Line: 6          Text: )
Token: NEWLINE          Line: 7          Text:

Token: PRINT            Line: 7          Text: PRINT
Token: L_PAR            Line: 7          Text: (
Token: STRING           Line: 7          Text: ""
Token: L_BRACK          Line: 7          Text: [
Token: COMMA            Line: 7          Text: ,
Token: ID               Line: 7          Text: x2
Token: R_BRACK          Line: 7          Text: ]
Token: R_PAR            Line: 7          Text: )
Token: QM               Line: 7          Text: ;
Token: NEWLINE          Line: 8          Text:

Token: ELSE             Line: 8          Text: ELSE
Token: NEWLINE          Line: 9          Text:

Token: PRINT            Line: 9          Text: PRINT
Token: L_PAR            Line: 9          Text: (
Token: STRING           Line: 9          Text: "The two numbers ar
1"
Token: R_PAR            Line: 9          Text: )
Token: QM               Line: 9          Text: ;
Token: NEWLINE          Line: 10         Text:

Token: ENDIF            Line: 10         Text: ENDIF
Token: NEWLINE          Line: 11         Text:

Token: RETURN           Line: 11         Text: RETURN
Token: ID               Line: 11         Text: x1
Token: QM               Line: 11         Text: ;
Token: NEWLINE          Line: 12         Text:

Token: END_FUNCTION             Line: 12          Text: END_FUNCTIO
Token: NEWLINE          Line: 13         Text:


Token: COMMENT          Line: 13         Text: %function/while test
```

```
Token: L_PAR            Line: 9          Text: (
Token: STRING           Line: 9          Text: "The two numbers ar
1"
Token: R_PAR            Line: 9          Text: )
Token: QM               Line: 9          Text: ;
Token: NEWLINE          Line: 10         Text:

Token: ENDIF            Line: 10         Text: ENDIF
Token: NEWLINE          Line: 11         Text:

Token: RETURN           Line: 11         Text: RETURN
Token: ID               Line: 11         Text: x1
Token: QM               Line: 11         Text: ;
Token: NEWLINE          Line: 12         Text:

Token: END_FUNCTION             Line: 12          Text: END_FUNCTIO
Token: NEWLINE          Line: 13         Text:


Token: COMMENT          Line: 13         Text: %function/while tes

Token: FUNCTION         Line: 13         Text: FUNCTION
Token: ID               Line: 13         Text: doStuff
Token: L_PAR            Line: 13         Text: (
Token: ID               Line: 13         Text: INTEGER
Token: ID               Line: 13         Text: var1
Token: COMMA            Line: 13         Text: ,
Token: ID               Line: 13         Text: INTEGER
Token: ID               Line: 13         Text: var2
Token: R_PAR            Line: 13         Text: )
Token: NEWLINE          Line: 14         Text:

Token: WHILE            Line: 14         Text: WHILE
Token: L_PAR            Line: 14         Text: (
Token: ID               Line: 14         Text: var1
Token: LT               Line: 14         Text: <
Token: INTEGER          Line: 14         Text: 10
Token: AND_OP           Line: 14         Text: AND
Token: ID               Line: 14         Text: var2
Token: LT               Line: 14         Text: <
Token: INTEGER          Line: 14         Text: 20
Token: R_PAR            Line: 14         Text: )
Token: NEWLINE          Line: 15         Text:

Token: IF               Line: 15         Text: IF
Token: L_PAR            Line: 15         Text: (
Token: ID               Line: 15         Text: var1
Token: EQ_OP            Line: 15         Text: ==
Token: ID               Line: 15         Text: var2
Token: R_PAR            Line: 15         Text: )
Token: THEN             Line: 15         Text: THEN
Token: NEWLINE          Line: 16         Text:

Token: BREAK            Line: 16         Text: BREAK
Token: QM               Line: 16         Text: ;
Token: NEWLINE          Line: 17         Text:

Token: ENDIF            Line: 17         Text: ENDIF
Token: NEWLINE          Line: 18         Text:

Token: ID               Line: 18         Text: var1
Token: ASSIGN           Line: 18         Text: =
Token: INTEGER          Line: 18         Text: 45
Token: QM               Line: 18         Text: ;
Token: NEWLINE          Line: 19         Text:

Token: ID               Line: 19         Text: var2
Token: GE_OP            Line: 19         Text: >=
Token: INTEGER          Line: 19         Text: 56
Token: QM               Line: 19         Text: ;
Token: NEWLINE          Line: 20         Text:

Token: ENDWHILE         Line: 20         Text: ENDWHILE
Token: NEWLINE          Line: 21         Text:


Token: RETURN           Line: 21         Text: RETURN
Token: ID               Line: 21         Text: var2
Token: QM               Line: 21         Text: ;
Token: NEWLINE          Line: 22         Text:

Token: END_FUNCTION             Line: 22          Text: END_FUNCTIO
Token: NEWLINE          Line: 23         Text:
```

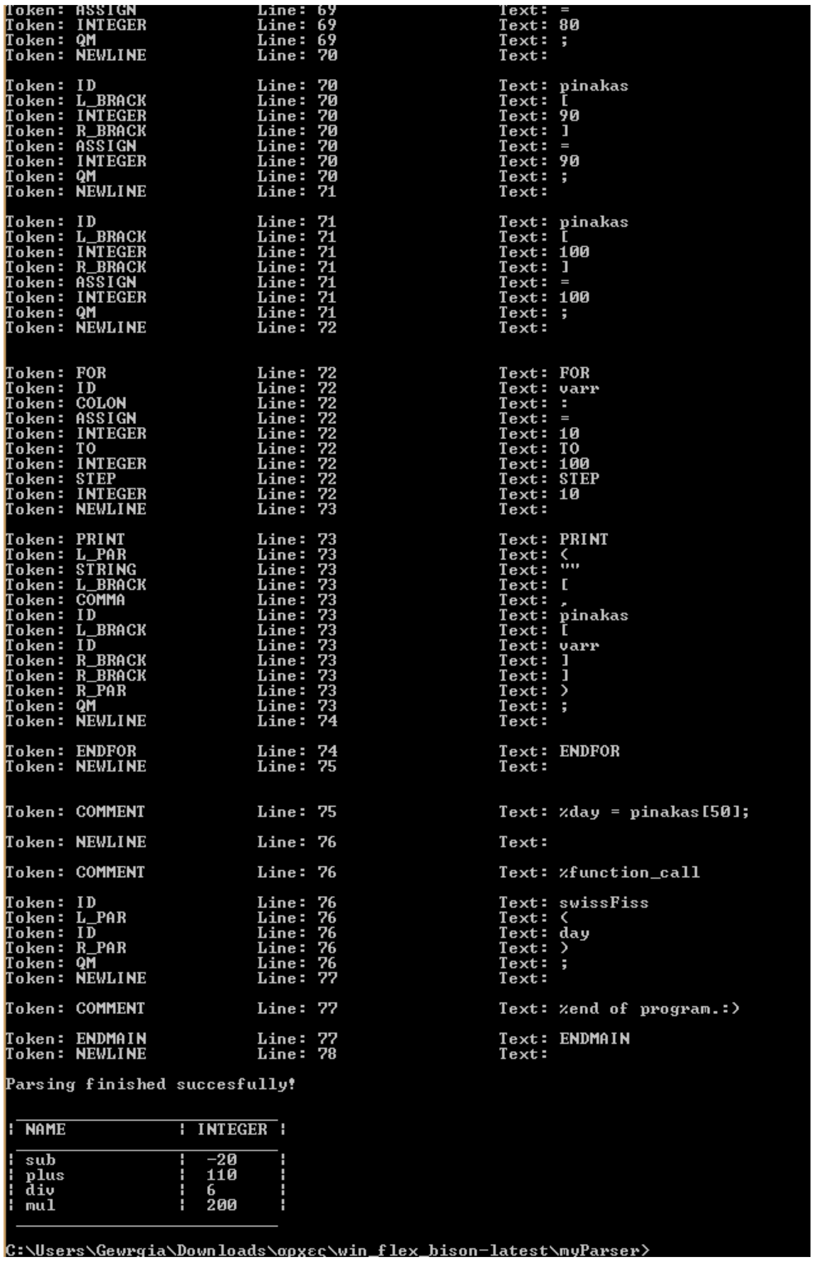
```
oken: INTEGER           Line: 41         Text: 16
oken: GT                Line: 41         Text: >
oken: R_PAR             Line: 41         Text: )
oken: NEWLINE           Line: 42         Text:

oken: PRINT             Line: 42         Text: PRINT
oken: L_PAR             Line: 42         Text: (
oken: STRING            Line: 42         Text: "Saturday"
oken: R_PAR             Line: 42         Text: )
oken: QM                Line: 42         Text: ;
oken: NEWLINE           Line: 43         Text:


oken: BREAK             Line: 43         Text: BREAK
oken: QM                Line: 43         Text: ;
oken: NEWLINE           Line: 44         Text:

oken: CASE              Line: 44         Text: CASE
oken: L_PAR             Line: 44         Text: (
oken: LT                Line: 44         Text: <
oken: INTEGER           Line: 44         Text: 17
oken: GT                Line: 44         Text: >
oken: R_PAR             Line: 44         Text: )
oken: NEWLINE           Line: 45         Text:

oken: PRINT             Line: 45         Text: PRINT
oken: L_PAR             Line: 45         Text: (
oken: STRING            Line: 45         Text: "Sunday"
oken: R_PAR             Line: 45         Text: )
oken: QM                Line: 45         Text: ;
oken: NEWLINE           Line: 46         Text:

oken: BREAK             Line: 46         Text: BREAK
oken: QM                Line: 46         Text: ;
oken: NEWLINE           Line: 47         Text:

oken: ENDSWITCH                 Line: 47          Text: ENDSWITCH
oken: NEWLINE           Line: 48         Text:

oken: RETURN            Line: 48         Text: RETURN
oken: ID                Line: 48         Text: day
oken: QM                Line: 48         Text: ;
oken: NEWLINE           Line: 49         Text:

oken: END_FUNCTION              Line: 49          Text: END_FUNCTION
oken: NEWLINE           Line: 50         Text:


oken: STARTMAIN                 Line: 50          Text: STARTMAIN
oken: NEWLINE           Line: 51         Text:

oken: VARS              Line: 51         Text: VARS
oken: NEWLINE           Line: 52         Text:

oken: ID                Line: 52         Text: CHAR
oken: ID                Line: 52         Text: char1
oken: COMMA             Line: 52         Text: ,
oken: ID                Line: 52         Text: char2
oken: QM                Line: 52         Text: ;
oken: NEWLINE           Line: 53         Text:

oken: ID                Line: 53         Text: INTEGER
oken: ID                Line: 53         Text: varr
oken: COMMA             Line: 53         Text: ,
oken: ID                Line: 53         Text: foo
oken: COMMA             Line: 53         Text: ,
oken: ID                Line: 53         Text: foo1
oken: COMMA             Line: 53         Text: ,
oken: ID                Line: 53         Text: foo2
oken: COMMA             Line: 53         Text: ,
oken: ID                Line: 53         Text: foo3
oken: COMMA             Line: 53         Text: ,
oken: ID                Line: 53         Text: foo4
oken: COMMA             Line: 53         Text: ,
oken: ID                Line: 53         Text: foo5
oken: COMMA             Line: 53         Text: ,
oken: ID                Line: 53         Text: pinakas
oken: L_BRACK           Line: 53         Text: [
oken: INTEGER           Line: 53         Text: 100
oken: R_BRACK           Line: 53         Text: ]
oken: COMMA             Line: 53         Text: ,
oken: ID                Line: 53         Text: day
oken: QM                Line: 53         Text: ;
```



```
Token: STARTMAIN                Line: 50          Text: STARTMAIN
Token: NEWLINE          Line: 51         Text:

Token: VARS             Line: 51         Text: VARS
Token: NEWLINE          Line: 52         Text:

Token: ID               Line: 52         Text: CHAR
Token: ID               Line: 52         Text: char1
Token: COMMA            Line: 52         Text: ,
Token: ID               Line: 52         Text: char2
Token: QM               Line: 52         Text: ;
Token: NEWLINE          Line: 53         Text:

Token: ID               Line: 53         Text: INTEGER
Token: ID               Line: 53         Text: varr
Token: COMMA            Line: 53         Text: ,
Token: ID               Line: 53         Text: foo
Token: COMMA            Line: 53         Text: ,
Token: ID               Line: 53         Text: foo1
Token: COMMA            Line: 53         Text: ,
Token: ID               Line: 53         Text: foo2
Token: COMMA            Line: 53         Text: ,
Token: ID               Line: 53         Text: foo3
Token: COMMA            Line: 53         Text: ,
Token: ID               Line: 53         Text: foo4
Token: COMMA            Line: 53         Text: ,
Token: ID               Line: 53         Text: foo5
Token: COMMA            Line: 53         Text: ,
Token: ID               Line: 53         Text: pinakas
Token: L_BRACK          Line: 53         Text: [
Token: INTEGER          Line: 53         Text: 100
Token: R_BRACK          Line: 53         Text: ]
Token: COMMA            Line: 53         Text: ,
Token: ID               Line: 53         Text: day
Token: QM               Line: 53         Text: ;
Token: NEWLINE          Line: 54         Text:

Token: NEWLINE          Line: 55         Text:

Token: COMMENT          Line: 55         Text: % calculations

Token: ID               Line: 55         Text: foo
Token: ASSIGN           Line: 55         Text: =
Token: FLOAT            Line: 55         Text: 10.8
Token: QM               Line: 55         Text: ;
Token: NEWLINE          Line: 56         Text:

Token: ID               Line: 56         Text: foo1
Token: ASSIGN           Line: 56         Text: =
Token: INTEGER          Line: 56         Text: 50
Token: QM               Line: 56         Text: ;
Token: NEWLINE          Line: 57         Text:

Token: COMMENT          Line: 57         Text: %foo2 = foo + foo1

Token: COMMENT          Line: 57         Text: %foo3 = foo * foo2;

Token: ID               Line: 57         Text: foo4
Token: ASSIGN           Line: 57         Text: =
Token: ID               Line: 57         Text: doStuff
Token: L_PAR            Line: 57         Text: (
Token: ID               Line: 57         Text: foo3
Token: COMMA            Line: 57         Text: ,
Token: ID               Line: 57         Text: foo1
Token: R_PAR            Line: 57         Text: )
Token: QM               Line: 57         Text: ;
Token: NEWLINE          Line: 58         Text:


Token: ID               Line: 58         Text: plus
Token: ASSIGN           Line: 58         Text: =
Token: INTEGER          Line: 58         Text: 50
Token: PLUS             Line: 58         Text: +
Token: INTEGER          Line: 58         Text: 60
Token: QM               Line: 58         Text: ;
Token: NEWLINE          Line: 59         Text:

Token: ID               Line: 59         Text: mul
Token: ASSIGN           Line: 59         Text: =
Token: INTEGER          Line: 59         Text: 10
Token: MUL              Line: 59         Text: *
Token: INTEGER          Line: 59         Text: 20
Token: QM               Line: 59         Text: ;
```

```
Token: ID               Line: 57         Text: foo4
Token: ASSIGN           Line: 57         Text: =
Token: ID               Line: 57         Text: doStuff
Token: L_PAR            Line: 57         Text: (
Token: ID               Line: 57         Text: foo3
Token: COMMA            Line: 57         Text: ,
Token: ID               Line: 57         Text: foo1
Token: R_PAR            Line: 57         Text: )
Token: QM               Line: 57         Text: ;
Token: NEWLINE          Line: 58         Text:


Token: ID               Line: 58         Text: plus
Token: ASSIGN           Line: 58         Text: =
Token: INTEGER          Line: 58         Text: 50
Token: PLUS             Line: 58         Text: +
Token: INTEGER          Line: 58         Text: 60
Token: QM               Line: 58         Text: ;
Token: NEWLINE          Line: 59         Text:

Token: ID               Line: 59         Text: mul
Token: ASSIGN           Line: 59         Text: =
Token: INTEGER          Line: 59         Text: 10
Token: MUL              Line: 59         Text: *
Token: INTEGER          Line: 59         Text: 20
Token: QM               Line: 59         Text: ;
Token: NEWLINE          Line: 60         Text:

Token: ID               Line: 60         Text: div
Token: ASSIGN           Line: 60         Text: =
Token: INTEGER          Line: 60         Text: 300
Token: DIV              Line: 60         Text: /
Token: INTEGER          Line: 60         Text: 50
Token: QM               Line: 60         Text: ;
Token: NEWLINE          Line: 61         Text:

Token: ID               Line: 61         Text: sub
Token: ASSIGN           Line: 61         Text: =
Token: INTEGER          Line: 61         Text: 40
Token: MINUS            Line: 61         Text: -
Token: INTEGER          Line: 61         Text: 60
Token: QM               Line: 61         Text: ;
Token: NEWLINE          Line: 62         Text:


Token: ID               Line: 62         Text: pinakas
Token: L_BRACK          Line: 62         Text: [
Token: INTEGER          Line: 62         Text: 10
Token: R_BRACK          Line: 62         Text: ]
Token: ASSIGN           Line: 62         Text: =
Token: INTEGER          Line: 62         Text: 10
Token: QM               Line: 62         Text: ;
Token: NEWLINE          Line: 63         Text:

Token: ID               Line: 63         Text: pinakas
Token: L_BRACK          Line: 63         Text: [
Token: INTEGER          Line: 63         Text: 20
Token: R_BRACK          Line: 63         Text: ]
Token: ASSIGN           Line: 63         Text: =
Token: INTEGER          Line: 63         Text: 20
Token: QM               Line: 63         Text: ;
Token: NEWLINE          Line: 64         Text:

Token: ID               Line: 64         Text: pinakas
Token: L_BRACK          Line: 64         Text: [
Token: INTEGER          Line: 64         Text: 30
Token: R_BRACK          Line: 64         Text: ]
Token: ASSIGN           Line: 64         Text: =
Token: INTEGER          Line: 64         Text: 30
Token: QM               Line: 64         Text: ;
Token: NEWLINE          Line: 65         Text:

Token: ID               Line: 65         Text: pinakas
Token: L_BRACK          Line: 65         Text: [
Token: INTEGER          Line: 65         Text: 40
Token: R_BRACK          Line: 65         Text: ]
Token: ASSIGN           Line: 65         Text: =
Token: INTEGER          Line: 65         Text: 40
Token: QM               Line: 65         Text: ;
Token: NEWLINE          Line: 66         Text:

Token: ID               Line: 66         Text: pinakas
Token: L_BRACK          Line: 66         Text: [
```

```
Token: R_BRACK          Line: 64         Text: ]
Token: ASSIGN           Line: 64         Text: =
Token: INTEGER          Line: 64         Text: 30
Token: QM               Line: 64         Text: ;
Token: NEWLINE          Line: 65         Text:

Token: ID               Line: 65         Text: pinakas
Token: L_BRACK          Line: 65         Text: [
Token: INTEGER          Line: 65         Text: 40
Token: R_BRACK          Line: 65         Text: ]
Token: ASSIGN           Line: 65         Text: =
Token: INTEGER          Line: 65         Text: 40
Token: QM               Line: 65         Text: ;
Token: NEWLINE          Line: 66         Text:

Token: ID               Line: 66         Text: pinakas
Token: L_BRACK          Line: 66         Text: [
Token: INTEGER          Line: 66         Text: 50
Token: R_BRACK          Line: 66         Text: ]
Token: ASSIGN           Line: 66         Text: =
Token: INTEGER          Line: 66         Text: 50
Token: QM               Line: 66         Text: ;
Token: NEWLINE          Line: 67         Text:

Token: ID               Line: 67         Text: pinakas
Token: L_BRACK          Line: 67         Text: [
Token: INTEGER          Line: 67         Text: 60
Token: R_BRACK          Line: 67         Text: ]
Token: ASSIGN           Line: 67         Text: =
Token: INTEGER          Line: 67         Text: 60
Token: QM               Line: 67         Text: ;
Token: NEWLINE          Line: 68         Text:

Token: ID               Line: 68         Text: pinakas
Token: L_BRACK          Line: 68         Text: [
Token: INTEGER          Line: 68         Text: 70
Token: R_BRACK          Line: 68         Text: ]
Token: ASSIGN           Line: 68         Text: =
Token: INTEGER          Line: 68         Text: 70
Token: QM               Line: 68         Text: ;
Token: NEWLINE          Line: 69         Text:

Token: ID               Line: 69         Text: pinakas
Token: L_BRACK          Line: 69         Text: [
Token: INTEGER          Line: 69         Text: 80
Token: R_BRACK          Line: 69         Text: ]
Token: ASSIGN           Line: 69         Text: =
Token: INTEGER          Line: 69         Text: 80
Token: QM               Line: 69         Text: ;
Token: NEWLINE          Line: 70         Text:

Token: ID               Line: 70         Text: pinakas
Token: L_BRACK          Line: 70         Text: [
Token: INTEGER          Line: 70         Text: 90
Token: R_BRACK          Line: 70         Text: ]
Token: ASSIGN           Line: 70         Text: =
Token: INTEGER          Line: 70         Text: 90
Token: QM               Line: 70         Text: ;
Token: NEWLINE          Line: 71         Text:

Token: ID               Line: 71         Text: pinakas
Token: L_BRACK          Line: 71         Text: [
Token: INTEGER          Line: 71         Text: 100
Token: R_BRACK          Line: 71         Text: ]
Token: ASSIGN           Line: 71         Text: =
Token: INTEGER          Line: 71         Text: 100
Token: QM               Line: 71         Text: ;
Token: NEWLINE          Line: 72         Text:
```



```
Token: ASSIGN            Line: 69                Text: =
Token: INTEGER           Line: 69                Text: 80
Token: QM                Line: 69                Text: ;
Token: NEWLINE           Line: 70                Text:

Token: ID                Line: 70                Text: pinakas
Token: L_BRACK           Line: 70                Text: [
Token: INTEGER           Line: 70                Text: 90
Token: R_BRACK           Line: 70                Text: ]
Token: ASSIGN            Line: 70                Text: =
Token: INTEGER           Line: 70                Text: 90
Token: QM                Line: 70                Text: ;
Token: NEWLINE           Line: 71                Text:

Token: ID                Line: 71                Text: pinakas
Token: L_BRACK           Line: 71                Text: [
Token: INTEGER           Line: 71                Text: 100
Token: R_BRACK           Line: 71                Text: ]
Token: ASSIGN            Line: 71                Text: =
Token: INTEGER           Line: 71                Text: 100
Token: QM                Line: 71                Text: ;
Token: NEWLINE           Line: 72                Text:


Token: FOR               Line: 72                Text: FOR
Token: ID                Line: 72                Text: varr
Token: COLON             Line: 72                Text: :
Token: ASSIGN            Line: 72                Text: =
Token: INTEGER           Line: 72                Text: 10
Token: TO                Line: 72                Text: TO
Token: INTEGER           Line: 72                Text: 100
Token: STEP              Line: 72                Text: STEP
Token: INTEGER           Line: 72                Text: 10
Token: NEWLINE           Line: 73                Text:

Token: PRINT             Line: 73                Text: PRINT
Token: L_PAR             Line: 73                Text: (
Token: STRING            Line: 73                Text: ""
Token: L_BRACK           Line: 73                Text: [
Token: COMMA             Line: 73                Text: ,
Token: ID                Line: 73                Text: pinakas
Token: L_BRACK           Line: 73                Text: [
Token: ID                Line: 73                Text: varr
Token: R_BRACK           Line: 73                Text: ]
Token: R_BRACK           Line: 73                Text: ]
Token: R_PAR             Line: 73                Text: )
Token: QM                Line: 73                Text: ;
Token: NEWLINE           Line: 74                Text:

Token: ENDFOR            Line: 74                Text: ENDFOR
Token: NEWLINE           Line: 75                Text:


Token: COMMENT           Line: 75                Text: %day = pinakas[50];

Token: NEWLINE           Line: 76                Text:

Token: COMMENT           Line: 76                Text: %function_call

Token: ID                Line: 76                Text: swissFiss
Token: L_PAR             Line: 76                Text: (
Token: ID                Line: 76                Text: day
Token: R_PAR             Line: 76                Text: )
Token: QM                Line: 76                Text: ;
Token: NEWLINE           Line: 77                Text:

Token: COMMENT           Line: 77                Text: %end of program.:)

Token: ENDMAIN           Line: 77                Text: ENDMAIN
Token: NEWLINE           Line: 78                Text:

Parsing finished succesfully!

  __________________________________
¦ NAME              ¦ INTEGER ¦
  __________________________________
¦ sub               ¦   -20   ¦
¦ plus              ¦   110   ¦
¦ div               ¦   6     ¦
¦ mul               ¦   200   ¦
  __________________________________

C:\Users\Gewrgia\Downloads\αρχες\win_flex_bison-latest\myParser>
```