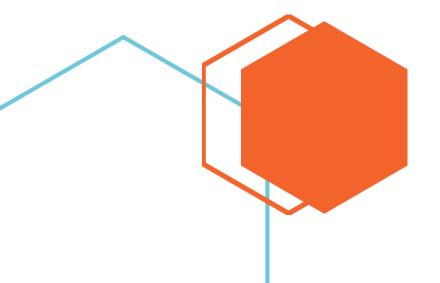


# ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

**Εργασία Flex & Bison** 

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







## ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ

## **Εργασία Flex & Bison**

Περιεχόμενα	
ΕΙΣΑΓΩΓΗ	
Περιγραφή γραμματικής της γλώσσας σε BNF	
Αρχείο BNF	
Ολοκληρωμένος Κώδικας	
Αρχείο Lexer.l	
Αρχείο Parser.y	
Αρχείο print_console.c	
Αρχείο εισαγωγής ψευδογλώσσας (input.cme)	
Παράδειγμα Εκτέλεσης	
If function:	
If function μέσα σε function:	
While:	
While in function:	
Switch Case:	
Switch Case in Function:	
<b>Vars</b> : (Με δήλωση Chars και Integers)	
<b>FOR</b> : (Με δήλωση πίνακα)	
Παράδειγμα Main Function:	
Έλεγχος λαθών	
Αποτέλεσμα με πράξεις (αποθήκευση τιμής σε μεταβλητή):	
Αποτελέσματα του αρχείου input.cme:	

#### 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 ΒΝΕ

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> | <for_statement> | <function> NEW
LINE
          | <function call>
           <comments> NEWLINE
          <variable>
           <switch>
          <while>
          <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>
                  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> <end if 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
                   | IDENTIFIER IDENTIFIER COMMA conditional_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
                  <inspector>::= IDENTIFIER <operators> <data_type>
         <data type> <operators> IDENTIFIER
<inspector gen> ::= <inspector>
             <comments> ::= COMMENT
<print> ::= PRINT L_PAR <data_type> R_PAR QM | PRINT L_PAR <data_type> <print_name_v</pre>
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
         <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
```

#### ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

## 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();
    void yyerror();
    //print statement function
    void print return(char *token);
%}
%x ML COMMENT
alphabet
                [a-zA-Z]
alphanumeric
                {alphabet}|{integer}
print
                [ -~]
underscore
                ({alphabet}|{underscore})+({alphanumeric}|{underscore})*
identifier
integer
                [0-9][0-9]*
                "0"|{integer}*"."{integer}+
float_number
char
                \'{print}\'
                \".*\"
string
WHITESPACE
                  [ \t]*
NEWLINE
                  [\n]*
%%
                 { print_return("COMMENT"); return COMMENT; }
```

```
<INITIAL>"/*"
                            { BEGIN(ML COMMENT); }
<ML_COMMENT>"*"+"/"
                         { BEGIN(INITIAL);}
<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; }
               { print_return("STARTMAIN"); return STARTMAIN; }
"STARTMAIN"
'ENDMAIN"
               { print_return("ENDMAIN"); return ENDMAIN;}
"IF"
               { print_return("IF"); return IF; }
               { print_return("THEN"); return THEN;}
"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; }
               { print_return("WHILE"); return WHILE;}
"WHILE"
               { print_return("ENDWHILE"); return ENDWHILE;}
"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; }
                { print_return("CHAR"); return CHAR; }
{char}
               { print_return("STRING"); return STRING; }
{string}
               {line no++; print return("NEWLINE"); return NEWLINE;}
{NEWLINE}
{WHITESPACE}
                {}
%%
     C FUNCTIONS
```

#### ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

```
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;
   extern FILE *yyout;
   extern int line no;
   //reads the input stream generates tokens
   extern int yylex();
   //temporary token save
   extern char* yytext;
   int yylex();
   void yyerror(char *message);
%}
        char name[500];
        int integer_val;
%token COMMENT
%token ML_COMMENT
%token BREAK
%token VARS
%token QM
%token NEWLINE
%token STARTMAIN
%token ENDMAIN
%token IF
%token THEN
```

#### ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

```
%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 QM NEWLINE ;
/*----*/
data_type: CHAR
       | INTEGER
       | FLOAT
       | IDENTIFIER
       | 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
/*----*/
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
/*----*/
inspector:IDENTIFIER operators data_type
       |data_type operators IDENTIFIER
inspector_gen: inspector
             | inspector operators
             | inspector operators inspector
             | inspector_gen QM
/*----*/
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: 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
/*----*/
comments: COMMENT | comments line | ml_comments;
ml_comments: ML_COMMENT ;
```

```
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
/*----*/
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 */
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 apχεία parser.y και lexer.l (-gcc lex.yy.c parser.tab.c) στον lexer εισάγεται το apχείο **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 apχείο 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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>. */
/* 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
```

```
#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,
     T0 = 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 ();
```

#### ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

```
#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** 

```
#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("_
    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 apχείο 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
/*-----PLEX AND BISON PROJECT -----
-----LEXICAL AND SYNTAX ANALYSIS ------
    ------HERE'S OUR PROGRAM------
   %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:

```
| Carrier | Carr
```

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

Κώδικας στη C στον οποίο οφείλεται:

```
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:\llse	es\Geuraia\Douploa	de\mnv:	cowin fley hison-	latest'	\muPaveev\ \a test
Token:	PROGRAM	Line:	eg\win_flex_bison= 1 1	Text:	PROGRAM
Token:	rs (dewrgia (Down 10a) PROGRAM ID NEWLINE	Line: Line:	1	Text: Text:	test
		rine.	2		
Token:	FUNCTION	Line:	2		FUNCTION
Token: Token:	I D	Line: Line:	0	Text: Text:	smaller
Token:	ID ID	Line:	2	Text:	INTEGER
Token: Token: Token:	ĪD	Line:	2	Text:	ÎNTEGER ×1
Token:	ÇOMMA	Line:	2	Text:	ÍNTEGER
Token: Token:	ID ID	Line:	2	Text: Text:	
Token:	R_PAR	Line: Line: Line: Line: Line: Line: Line:	2	Text:	
Token:	NEWLINE	Line:	3	Text:	
Token:	IF	Line:	3	Text:	IF
Token: Token:	L_PAR	Line: Line:	3	Text:	(
Token: Token:	I D	Line:	3	Text: Text:	x1 <
Token:	ID	Line: Line: Line: Line:	3	Text:	×2
Token: Token: Token:	R_PAR	Line:	3	Text:	>
Token:	THEN NEWLINE	Line:	3	Text:	THEN
		Line:	7	Text:	
Token:	PRINT L_PAR STRING L_BRACK	Line:	4	Text:	PRINT
Token:	L_PAR CTRING	Line: Line:	4	Text: Text:	( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
Token:	L BRACK	Line:	4	Text:	Ī.
		Line:	4	Text:	,
Token:	I D	Line:	4	Text:	
Token:	ID R_BRACK R_PAR QM	Line:	4	Text: Text:	1
Token:	QM	Line:	4	Text:	į
Token:	NEWLINE	Line:	5	Text:	
Token:	ELSEIF	Line:	5	Text:	ELSEIF
Token:	ELSEIF L_PAR ID	Line:	5	Text:	(
Token: Token:	ID	Line: Line:	5 5	Text:	×1 <
Token:	I D	Line:	5	Text: Text:	
Token: Token:	Ř_PAR	Line:	5	Text:	
Token:	NEWLINE	Line:	6	Text:	
Token:	PRINT	Line:	6	Text:	PRINT
Token: Token:	L_PAR	Line:	6	Text:	<
Token:	STRING	Line:	6	Text:	
Token:	COMMA	Line: Line:	6	Text: Text:	
Token:	ID	Line:	6	Text:	×2
Token:	R_BRACK	Line: Line:	6	Text:	1
Token:	M M	Line:	6	Text: Text:	,
Token:	L_PAR STRING L_BRACK COMMA ID R_BRACK R_PAR QM NEWLINE	Line: Line:	ž	Text:	
Token:		Line:	7	Text:	FICE
Token:	NEWLINE	Line:	8	Text:	ELSE
				_	
Token:	PRINT	Line:	8	Text:	PRINT
Token:	PRINT L_PAR STRING	Line: Line:	8	Text:	"The two numbers are equa
T.V					
Token: Token:	K_PHK	Line: Line:	8	Text: Text:	
Token:	NEWLINE	Line:	9	Text:	,
	F11777				T
Token:	NEWLINE	Line: Line:	10	Text:	ENDIF
Token:	RETURN	Line: Line: Line:	10	Text:	RETURN
Token: Token:	OM	Line:	10	Text: Text:	×1 ;
Token:	NEWLINE	Line:	īī	Text:	
Tolera	END PUNCTION		Line: 11		Tout: END PHACTION
Token:	END_FUNCTION NEWLINE	Line:	Line: 11 12	Text:	Text: END_FUNCTION
Parsing	g finished succesf	ılly!			

#### While:

WHILE(var1<10 AND var2<20)

IF(var1==var2) THEN

BREAK;

**ENDIF** 

var1 = 45;

var2 >= 56;

ENDWHILE

C:\usei	rs\Gewrgia\Downioa	ας /αρχ:	eç\win_flex_bison-	Latest	\myrarser/.\a test
Token:	PROGRAM	rine:	<u> </u>		PROGRAM
Token:	PROGRAM ID NEWLINE	Line:	1	Text:	
loken:	NEWLINE	Line:	Z	Text:	
T - 1	UULTE	T 4	0	T	LINETE TO
Token:	WHILE	rine:	2		ÄHITE
Loken:	r <sup>≝</sup> ьнк	rine:	Z	Text:	
Loken:	L_PAR ID LT	Line:	2	Text:	
loken:	WHILE L_PAR ID LT INTEGER AND_OP ID	rine:	Z	Text:	
Loken:	INTEGER	Line:	2	Text:	
loken:	AND_OP	Fine:	Z	Text:	
loken:	ID LT	Fine:	2	Text:	
loken:	LI	Line:	2	Text:	
loken:	INTEGER	Fine:	2	Text:	
Token:	R_PAR NEWLINE	Line:	2	Text:	
Token:	NEWLINE	Line:	3	Text:	
		Line: Line: Line: Line: Line: Line: Line: Line:	•	m .	
Token:	IF L_PAR	Line: Line: Line: Line: Line:	3	Text:	
Token:	L_PAR	Line:	3	Text:	
	ID	Line:	3	Text:	
Token:	EQ_OP ID	Line:	3	Text:	
Token:	ID	Line:	3	Text:	
Token:	R_PAR	Line: Line: Line:	3	Text:	
Token:	THEN	Line:	3	Text:	
Token:	NEWLINE	Line:	4	Text:	
	PPF411				PPP444
Token:	BREAK QM	Line:			BREAK
loken:	Qm	Line:		Text:	
Token:	NEWLINE	Line:	5	Text:	
T - 1 •	EMBLE	T 2	r.	T 4 -	EMBLE
Token:	NEWLI NE	Line: Line:		Text:	
Token:	NEWLINE	rine:	ь	Text:	
T - 1	TD	T 4		T	4
Token:	ID	rine:	þ	Text:	
Token:	H551GN	rine:	þ	Text:	
Loken:	INTEGER	rine:	þ	Text:	
Token:	ID ASSIGN INTEGER QM NEWLINE	rine:	b	Text:	
roken:	NEWLINE	Line:	7	lext:	
Tokon	ID	Line:	7	Text:	
TT 1	OF OR	Line:	<u></u>	Text:	
Token:	INTEGER	Line: Line:	<u>,</u>	Text:	
Token:	QM	Line:	5		
		Line:		Text: Text:	9
Token:	NEWLINE	rine:	0	lext:	
Tokon:	ENDULLE	Line:	0	Toyt:	ENDWHILE
Token:	ENDWHILE NEWLINE	Line:	9	Text:	ENDMIT LE
roken.	NEWBINE	True -	,	IEXL.	
Paneine	finished succesf	u11ut			
rarsing	, i inisneu succesi	ully:			

#### **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

oken: oken:	ID NEWLINE	Line: Line:		Text: Text: Text:	PROGRAM tets
ken:	FUNCTION ID L_PAR ID ID COMMA ID ID ID REPAR ID ID ID ID ID ID ID R_PAR NEWLINE	Line: Line: Line: Line: Line: Line: Line: Line: Line:	2	Text:	FUNCTION doStuff
ken:	I DOD	Line:	2	Text:	dostuff
iken:	ID ID	Line:	2	Text:	INTEGER
ken:	ĪD	Line:	$\bar{2}$	Text:	var1
ken:	ÇOMMA	Line:	2	Text:	ÍNTEGER
)ken:  ken:	עז עז	Line:	2	Text:	INIEGER
ken:	R PAR	Line:	2	Text:	)
ken:	NEWLINE	Line:	3	Text:	
ken:	WHILE L_PAR ID LT LT INTEGER AND_OP ID LT INTEGER REPAR NEWLINE	Line: Line: Line: Line: Line:	3		ÄHITE
ken:	L_PAK	Line:	3	Text: Text:	
iken:	LT	Line:	3	Text:	<
ken:	INTEGER	Line:	3	Text:	10
ken:	AND_OP	Line:	3	Text:	AND
ken:	I.T	Line: Line: Line:	ქ ე	Text: Text: Text: Text:	var2
ken:	INTEGER	Line: Line: Line:	3	Text:	20
ken:	R_PAR	Line:	3	Text:	>
ken:	NEWLINE	Line:	4	Text:	
ken:	IF	Line:	4	Text:	ĮF
ken:	L_PHK	Line: Line:	4	Text:	uant.
ken:	EQ OP	Line	4	Text:	==
ken:	ĪĎ	Line: Line: Line:	4	Text:	uar2
ken:	R_PAR	Line:	4	Text:	) TUT!
ken:	IF L_PAR ID EQ_OP ID R_PAR THEN NEWLINE	Line: Line:	5	Text:	IHEN
		Line:	5	Text:	BREAK
ken:	BREAK QM	Line: Line:	5	Text:	;
ken:	NEWLINE	Line:	6	Text:	
ken: ken:	ENDIF NEWLINE	Line: Line:	6 7	Text: Text:	ENDIF
ken:	ID	Line:	7	Text:	var1
ken:	ID ASSIGN_	Line:	7	Text:	
ken:	INTEGER	Line:	7	Text:	45
ken:	ASSIGN INTEGER QM NEWLINE	Line: Line: Line: Line: Line:	7 8	Text: Text:	;
				Text:	var2
ken:	ID GE_OP	Line: Line:	8	Text:	>=
ken:	INTEGER QM	Line:	8	Text: Text:	56
ken:	NEWLI NE	Line: Line: Line:	9	Text:	
ken:	ENDWHILE NEWLINE	Line: Line:	9	Text:	ENDWHILE
ken:	NEWLINE	Line:	10	Text:	
ken:	RETURN	Line:	10	Text:	RETURN
ken:	RETURN I D QM	Line:	10	Text:	var2
ken: ken:	QM NEWLINE	Line: Line: Line: Line:	10 11	Text: Text:	
ken:		Line:		Text:	Text: END_FUNCTION
	NEWLINE	Line:		Text:	
Mell -	MENDINE	TILE .	13	ICAU.	

Sw	•1	$\hat{}$	~-		
	4176	C	М	œ.	-
$\sim M_{\odot}$	-	 •	ч	<b>න</b>	u

SWITCH( <day>)</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

#### ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

en: PROGRAM	pads\σρχες\win_flex_b Line: 1 Line: 1 Line: 2	Text:	PROGRAM		R_PAR NEWLINE	Line: 12 Line: 13	Text:	>
en: NEWLINE	Line: 2	Text:		Token:	PRINT L_PAR STRING R_PAR QM NEWLINE	Line: 13 Line: 13 Line: 13 Line: 13 Line: 13	Text:	PRINT
en: SWITCH	Line: 2	Tevt:	SWITCH	Token:	L_PAR	Line: 13	Text:	PRINI { "Thursday" }
en: SWITCH en: L_PAR en: LT en: ID	Line: 2	Text: Text: Text:	(	Token:	STRING	Line: 13	Text:	"Thursday"
n: LT	Line: 2	Text:	<	Token:	K_PHK	Line: 13	Text:	?
n: ID	Line: 2	Text:	day	Token-	MELIT. I ME	Line: 13	Text:	,
n: GT n: R_PAR n: NEWLINE	Line: 2 Line: 2 Line: 2 Line: 2 Line: 2 Line: 2 Line: 3	Text: Text:	<	1011011-	112112112		10,00	
: NEWLINE	Line: 3	Text:	•	Token:	BREAK	Line: 14	Text:	BREAK
				Token:	QM NEWLINE	Line: 14 Line: 14 Line: 15	Text:	
n: CHSE	Line: 3 Line: 3 Line: 3	Text: Text: Text:	CHSE				Text:	
n: LT	Line: 3	Text:	2	Tokon -	COSE	Line: 15	Text:	COCE
n: INTEGER	Line: 3	Text:	11	Token:	L PAR	Line: 15	Text:	(
n: GT	Line: 3	Text: Text:	>	Token:	LT	Line: 15	Text:	<
n: CASE n: L PAR n: LT n: INTEGER n: G n: R PAR n: NEWLINE	Line: 3 Line: 3 Line: 3 Line: 4	Text	,	Token:	INTEGER	Line: 15	Text:	15
II- NEWBINE				Token:	GT	Line: 15	Text:	>
n: PRINT	Line: 4 Line: 4 Line: 4 Line: 4	Text:	PRINT	Token:	CASE L_PAR LT INTEGER GT R_PAR NEWLINE	Line: 15 Line: 15 Line: 15 Line: 15 Line: 15 Line: 15 Line: 15	Text:	,
n: L_PAK	Line: 4	Text:	''Monday''	TOVCII-	NEWBINE		ICAL.	
n: R PAR	Line: 4	Text:	)	Token:	PRINT	Line: 16	Text:	PRINT
n: QM	Line: 4	Text:	;	Token:	L_PAR	Line: 16	Text:	PRINT
n: PRINT n: L_PAR n: STRING n: R_PAR n: QM n: NEWLINE	Line: 5	Text:		Token:	STRING	Line: 16	Text:	"Friday" >
n: BREAK	Line: 5	Text:	BBEOK	loken:	K_PAK	Line: 16	Text:	?
n: QM n: NEWLINE	Line: 5 Line: 5 Line: 6	Text:	;	Token:	PRINT L_PAR STRING R_PAR QM NEWLINE	Line: 16 Line: 16 Line: 16 Line: 16 Line: 16 Line: 17	Text: Text:	
: NEWLINE	Line: 6	Text:					1676-	
. CASE	Line: 6	Text:	COSE	Token:	BREAK QM NEWLINE	Line: 17 Line: 17 Line: 18		BREAK
n: CASE n: L_PAR n: LT n: INTEGER	Line: 6	Text	(	Token:	QM	Line: 17	Text:	
n: LT	Line: 6 Line: 6 Line: 6	Text: Text: Text:	<_	loken:	NEWLINE	Line: 18	Text:	
: INTEGER	Line: 6	Text:	12	Token	COSE	Line: 18	Text:	CASE
: R PAR	Line: 6	Text: Text:	<u> </u>	Token:	L PAR	Line: 18 Line: 18	Text:	(
: GT : R_PAR : NEWLINE	Line: 6 Line: 6 Line: 7	Text		Token: Token: Token:	LT		Text:	<
			TOTAL	Token:	INTEGER GT R_PAR NEWLINE	Line: 18 Line: 18 Line: 18 Line: 18 Line: 19	Text:	16
FRINT	Line: 7 Line: 7 Line: 7 Line: 7 Line: 7 Line: 8	Text:	PRINT < "Tuesday" >	Token:	GT	Line: 18	Text:	>
: STRING	Line: 7	Text	"Tuesday"	loken:	K_PHK	Line: 18	Text: Text:	,
: PRINT : L_PAR : STRING : R_PAR : QM : NEWLINE	Line: 7	Text:	>				I EXT -	
: QM	Line: 7	1ext:	;	Token:	PRINT	Line: 19 Line: 19 Line: 19 Line: 19 Line: 19	Text:	PRINT
	riue: 8	Text:		Token:	L_PAR	Line: 19	Text:	(
n: BREAK n: QM n: NEWLINE	Line: 8	Text:	BREAK	Token:	STRING	Line: 19	Text:	PRINT { "Saturday" }
i: QM	Line: 8 Line: 9	Text:	;	loken:	K_PHK	Line: 19	Text: Text:	?
: NEWLINE		Text:		Token:	PRINT L_PAR STRING R_PAR QM NEWLINE	Line: 19 Line: 20	Text:	
n: CASE	Line: 9	Text:	CASE	TORUIT-		2200 - 20	IUXU.	
ı: L_PAR	Line: 9	Text:	(					
n: LI	Line: 9	Text: Text:	13	Token:	BREAK	Line: 20	Text:	BREAK
n: CHSE n: L.PAR n: LT n: INTEGER n: INTEGER n: R_PAR n: NEWLINE	Line: 9	Text	>	Token:	QM NEWLINE	Line: 20 Line: 20 Line: 21	Text:	,
n: R_PAR	Line: 9 Line: 9 Line: 9 Line: 9 Line: 9 Line: 9 Line: 10	Text: Text:	>				IEXU-	
: NEWLINE	Line: 10	Text:		Token:	CASE	Line: 21	Text:	CASE
: PRINT		Text:	PRINT	Token:	L_PAR	Line: 21	Text:	(
: L_PAR	Line: 10	Text	PRINT ( "Wednesday"	Token:	LI	Line: 21	Text	<b>4</b>
: STRING	Line: 10	Text:	"Wednesday"	loken:	INI EGER	Line: 21	Text: Text:	17
: K_PAK	Line: 10	Text: Text:		Token	R PAR	Line: 21	Text:	3
: PRINT : L_PAR : STRING : R_PAR : QM : NEWLINE	Line: 10 Line: 10 Line: 10 Line: 10 Line: 10 Line: 11	Text	,	Token:	CASE L_PAR LT INTEGER GT R_PAR NEWLINE	Line: 21 Line: 21 Line: 21 Line: 21 Line: 21 Line: 21 Line: 22	Text:	
			PPEAN					
: BREAK : QM : NEWLINE	Line: 11 Line: 11 Line: 12	Text: Text:	BREHK	Token:	PRINT L_PAR STRING R_PAR QM NEWLINE	Line: 22 Line: 22 Line: 22 Line: 22 Line: 22 Line: 23	Text	PRINT
: NEWLINE	Line: 12	Text	,	Token:	L_FHK STRING	Line: 22	Text	"Sundau"
				Token	R PAR	Line: 22	Text	("Sunday"
: CASE	Line: 12	Text	CASE	Token:	QM	Line: 22	Text:	;
- L_ГНК : LT	Line: 12	Text: Text:		Token:	NEWLINE	Line: 23	Text:	
: INTEGER	Line: 12	Text:	14				Т	BBEAV
: GT	Line: 12	Text: Text:	>	loken:	BREAK	Line: 23	Text: Text:	BREAK
E CASE E LPAR E LT E INTEGER E R_PAR E NEWLINE	Line: 12 Line: 12 Line: 12 Line: 12 Line: 12 Line: 12 Line: 13	Text: Text:	>	Token:	QM NEWLINE	Line: 23 Line: 23 Line: 24	Text:	
- MENDINE							1000	
n: PRINT n: L_PAR n: STRING n: R_PAR n: QM	Line: 13 Line: 13	Text:	PRINT	Token:	ENDSWITCH	Line: 24 Line: 25	_	Text: ENDSWITCH
E L_PAK	Line: 13	Text:	"Thursday" );	Token:	NEWLINE	Line: 25	Text:	
			inar sua 9					
R_PAR	Line: 13 Line: 13 Line: 13	Text:	> -	Parein	g finished succe	efullu#		

## **Switch Case** in Function:

FUNCTION swissFiss (INTEGER day)
SWITCH( <day>)</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

#### ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

• •

Jsers\Gewrgia\Dowr en: PROGRAM	lloads\αρχες\win_flex Line: 1	<pre>c_bison-latest\myParser&gt;.\a test     Text: PROGRAM     Text: test</pre>	Token: PRINT Token: L_PAR Token: STRING	Line: 14 Line: 14 Line: 14	Text: PRINT Text: ( Text: "Thursday"
en: ID	Line: 1	Text: test	Token: STRING	Line: 14	Text: "Thursday"
en: NEWLINE	Line: 2	Text:	Token: R_PAR Token: QM Token: NEWLINE	Line: 14 Line: 14	Text: )
D. FUNCTION	Line: 2	Text: FUNCTION	Token: QM	Line: 14 Line: 15	Text: ;
en: FUNCTION en: ID en: L_PAR en: ID en: ID en: ID en: R_PAR en: NEWLINE	Line: 2 Line: 2 Line: 2	Text: swissFiss	Token: NEWLINE	Line: 15	Text:
en: L PAR	Line: 2	Text: (	Token: BREAK	Line: 15	Text: BREAK
en: ID	Line: 2	Text: INTEGER	Token: QM	Line: 15	Text: ;
en: ID	Line: 2 Line: 2	Text: day	Token: QM Token: NEWLINE	Line: 15 Line: 15 Line: 16	Text:
en: R_PAR	Line: 2	Text: >			
en: NEWLINE	Line: 3	Text:	Token: CASE	Line: 16	Text: CASE
	Line: 3	Text: SWITCH	Token: L_PAR Token: LT	Line: 16	Text: (
en: L PAR	Line: 3	Text: (	Token: INTEGER	Line: 16 Line: 16	Text: < Text: 15
en: SWITCH en: L_PAR en: LT en: ID	Line: 3 Line: 3	Text: < Text: <	Token: GT Token: R_PAR Token: NEWLINE	Line: 16	Text: >
en: ID	Line: 3	Text: day	Token: R_PAR	Line: 16	Text: > Text: >
en: GT en: R_PAR en: NEWLINE	Line: 3	Text: >	Token: NEWLINE	Line: 17	Text:
en: R_PAR	Line: 3	Text: >			
en: NEWLINE	Line: 4	Text:	Token: PRINT Token: L_PAR Token: STRING Token: R_PAR	Line: 17	Text: PRINT
en: CASE	Line: 4	Text: CASE	Ioken: L_PHK	Line: 17 Line: 17 Line: 17 Line: 17	Text: ( Text: "Friday"
en: L PAR	Line: 4	Text: (	Token: SIKING	Line: 17	Text: )
en: L_PAR en: LT en: INTEGER en: GT	Line: 4	Text: <	Token: A_rhh	Line: 17	Text: ;
en: INTEGER	Line: 4	Text: 11	Token: QM Token: NEWLINE	Line: 18	Text:
en: GT	Line: 4	Text: >	TORON- NEWBINE	H1110. 10	10.00
en: R_PAR en: NEWLINE	Line: 4	Text: )	Token: BREAK	Line: 18	Text: BREAK
en: NEWLINE	Line: 5	Text:	Token: QM Token: NEWLINE	Line: 18	Text: ;
DDINT	Line: E	Toyt: DRINT	Token: NEWLINE	Line: 19	Text:
D I PAR	Line: 5	Text: PRINT Text: <	T-1 CASE	13 10	T CARE
en: PRINT en: L_PAR en: STRING	Line: 5 Line: 5	Text: "Monday"	Token: CASE	Line: 19 Line: 19	Text: CASE
en: R_PAR	Line: 5	Text: )	Token: L_PAR Token: LT	Line: 19 Line: 19	Text: CHSE Text: ( Text: (
en: R_PAR en: QM en: NEWLINE	Line: 5 Line: 5	Text: ;		Line: 19	Text: 16
en: NEWLINE	Line: 6	Text:	Token: GT Token: R_PAR Token: NEWLINE	Line: 19 Line: 19	Text: >
			Token: R_PAR	Line: 19	Text: >
en: BREAK en: QM en: NEWLINE	Line: 6	Text: BREAK	Token: NEWLINE	Line: 20	Text:
en: UM	Line: 6 Line: 7	Text: ;			
	Line: /	Text:	Token: PRINT Token: L_PAR Token: STRING	Line: 20	Text: PRINT Text: (
n. COSE	Line: 7	Text: CASE	Ioken: L_PAK	Line: 20	Text: ( Text: "Saturday"
en: CASE en: L_PAR en: LT en: INTEGER en: GT	Line: 7 Line: 7 Line: 7 Line: 7 Line: 7	Text: (	Token: SIKING	Line: 20 Line: 20 Line: 20 Line: 20 Line: 20 Line: 21	Text: "Saturday" Text: >
en: LT	Line: 7	Text: <	Token: R_PAR Token: QM Token: NEWLINE	Line: 20	Text: ;
en: INTEGER	Line: 7	Text: 12	Token: NEULINE	Line: 21	Text:
en: GT	Line: 7	Text: >	101011- 112112112	22110 - 22	10/10-
en: R_PAR en: NEWLINE	Line: 7	Text: >			
en: NEWLINE	Line: 8	Text:	Token: BREAK	Line: 21 Line: 21 Line: 22	Text: BREAK
en: PRINT	Line: 8	Text: PRINT	Token: QM Token: NEWLINE	Line: 21	Text: ;
en: L PAR	Line: 8	Tayt: (	loken: NEWLINE	Line: 22	Text:
en: L_PAR en: STRING en: R_PAR	Line: 8 Line: 8	Text: "Tuesday" Text: >	Token: CASE	Line: 22	Text: CASE
en: R PAR	Line: 8	Text: )	Token: L PAP	Line: 22	Text: CASE Text: ( Text: <
en: QM en: NEWLINE	Line: 8	Text: ;	Token: L_PAR Token: LT	Line: 22	Text: <
en: NEWLINE	Line: 9	Text:	Token: INTEGER	Line: 22	Text: 17
- DDEAK		T DDFAV	Token: GT	Line: 22	Text: >
en: BREAK	Line: 9 Line: 9	Text: BREAK	Token: R_PAR	Line: 22	Text: >
en: QM en: NEWLINE	Line: 9 Line: 10	Text: ; Text:	Token: R_PAR Token: NEWLINE	Line: 22 Line: 22 Line: 22 Line: 22 Line: 22 Line: 22 Line: 23	Text:
	TILE - 18	16.40			T DRINT
en: CASE	Line: 10	Text: CASE	Token: PRINT Token: L_PAR	Line: 23	Text: PRINT Text: (
en: CASE en: L_PAR en: LT en: INTEGER	Line: 10 Line: 10	Text: (	Token: L_PHK Token: STRING	Line: 23	Text: ( Text: "Sunday"
en: LT	Line: 10	Text: <	Token: R PAR	Line: 23 Line: 23 Line: 23 Line: 23 Line: 23 Line: 24	Text: >
en: INTEGER	Line: 10	Text: 13	Token: R_PAR Token: QM Token: NEWLINE	Line: 23	Text: ;
en: GT en: R_PAR en: NEWLINE	Line: 10	Text: >	Token: NEWLINE	Line: 24	Text:
en = K_PHK	Line: 10 Line: 11	Text: >			
	Line: 11	Text:	Token: BREAK	Line: 24 Line: 24 Line: 25	Text: BREAK
en: PRINT	Line: 11	Text: PRINT	Token: QM Token: NEWLINE	Line: 24	Text: ;
en: L_PAR	Line: 11	Text: (	TOKEN: NEWLINE	Line: Z5	Text:
en: PRINT en: L_PAR en: STRING en: R_PAR en: QM en: NEWLINE	Line: 11	Text: "Wednesday"	Token: ENDSWITCH	Line: 25	Text: ENDSWITCH
en: R_PAR	Line: 11	Text: )	Token: NEWLINE	Line: 26	Text:
en: QM	Line: 11	Text: ;	TOTAL TIENDATIO		
en: NEWLINE	Line: 12	Text:	Token: RETURN	Line: 26	Text: RETURN
en: BREAK	Line: 12	Text: BREAK	Token: ID	Line: 26 Line: 26 Line: 26 Line: 27	Text: day
en: BKEHK en: QM	Line: 12 Line: 12	Text: BREHK	Token: QM	Line: 26	Text:;
en: NEWLINE	Line: 13	Text:	Token: NEWLINE	Line: 27	Text:
			Tokon: END FUNCTION	Line: 27	Touts END PUNCTI
en: CASE	Line: 13	Text: CASE	Token: END_FUNCTION Token: NEWLINE	Line: 28	Text: END_FUNCTION
en: CASE en: L_PAR en: LT	Line: 13 Line: 13 Line: 13	Text: (	TOKEH- MEMBINE	Hale. 20	IUXU-
en: LT	Line: 13	Text: <			
	Line: 13	Text: 14	Danadan Ciniahad assau	nofullut	
en: INTEGER	22110 - 20		Parsing finished succ		

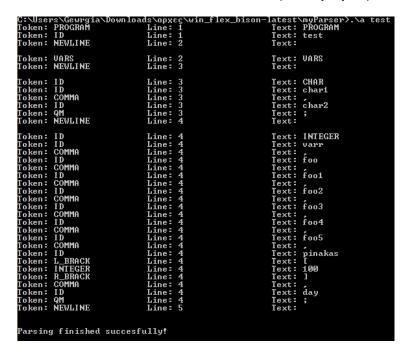
Επίσης, για τις συναρτήσεις βλέπουμε ότι υποχρεωτικά πριν το END\_FUNCTION πρέπει να υπάρχει RETURN αλλιώς εμφανίζεται μήνυμα συντακτικού λάθους.

#### Vars: (Με δήλωση Chars και Integers)

**VARS** 

CHAR char1, char2;

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



#### **FOR**: (Με δήλωση πίνακα)

FOR varr:=10 TO 100 STEP 10

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

ENDFOR

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:

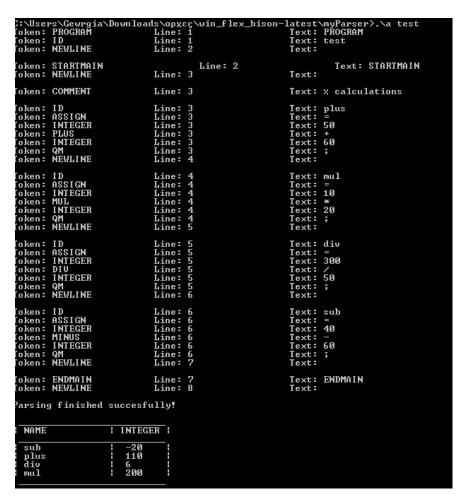
% calculations plus = 50 + 60; mul = 10\*20;

STARTMAIN

div = 300/50;

sub = 40-60;

**ENDMAIN** 



#### 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.

```
ID
NEWLINE
oken: NEWLINE
                                                                         Text:
oken: COMMENT
                                                                         Text: %function/if test
        FUNCTION
        ID
L_PAR
ID
ID
COMMA
                                                                                  ÎNTEGER
                                                                                  ÍNTEGER
×2
>
        R_PAR
NEWLINE
        IF
L_PAR
ID
LT
ID
R_PAR
THEN
NEWLINE
        QM
NEWLINE
       ELSEIF
L_PAR
ID
LT
ID
        R_PAR
NEWLINE
        ID
R_BRACK
R_PAR
        QM
NEWLINE
oken: ELSE
oken: NEWLINE
                                                                         Text: (
Text: "The two numbers are equ
                                                                        Text: >
Text:
ror: "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
Token: ID
Token: NEWLINE
                                                                                                PŔOGRAM
                                                                                      Text:
Text:
Text:
                                            Line:
Line:
                                                                                                test
Token: FUNCTION
Token: ID
                                            Line:
                                                                                      Text: FUNCTION
Token: FONCI
Token: ID
Token: ID
Token: ID
Token: COMMA
                                            Line:
                                                                                                error
                                            Line:
Line:
Line:
                                                                                      Text: ÎNTEGER
                                                                                      Text: error1
                                            Line:
                                                                                      Text: ÍNTEGER
Text: error?
Token: ID
Token: ID
Token: R_PAR
Token: NEWLINE
                                            Line:
Line:
                                                                                                error2
                                            Line:
                                                                                      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:

```
C:\Users\Gewrgia\Downloads\opxec\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".

ain function			
	Token: COMMENT	Line: 74	Text: %day = pinakas[50];
ARTMAIN			
VARS	Token: NEWLINE	Line: 75	Text:
CHAR char1, char2;		221121 73	rexer
INTEGER varr, foo, foo1, foo2, foo3, foo	Token: COMMENT	Line: 75	Text: %function_call
% calculations	T 1 TO		T
foo = 10;	Token: ID	Line: 75	Text: swissFiss
kati = 100;	Token: L_PAR	Line: 75	Text: (
foo2 = 10 + 100 + 10;	Token: ID	Line: 75	Text: day
foo3 = foo2 + 30;	Token: R_PAR	Line: 75	Text: )
<pre>foo4 = doStuff(foo3, foo1);</pre>	Token: QM	Line: 75	Text: ;
	Token: NEWLINE	Line: 76	Text:
pinakas[10] = 10;			
pinakas[20] = 20;	Token: COMMENT	Line: 76	Text: %end of program.:)
pinakas[30] = 30;			
pinakas[40] = 40;	Token: ENDMAIN	Line: 76	Text: ENDMAIN
pinakas[50] = 50;	Token: NEWLINE	Line: 77	Text:
pinakas[60] = 60;			
pinakas[70] = 70;	Parsing finished	succesfully!	
pinakas[80] = 80;	9		
pinakas[90] = 90;			
pinakas[100] = 100;	NAME	INTEGER	
FOR varr:=10 TO 100 STEP 10	foo2	120	
PRINT(""[,pinakas[varr]]);	foo3	150	
ENDFOR	1003	1 230	
Yday - ninakas[[0]]			
%day = pinakas[50];	F:\School\2020 P	rojects\Arxes BNF-mai	n>
%function_call	2. (5611001 (2020_1	rojeces (ili xes_biti lilat	
ኤተunction_call swissFiss(day);			

## Αποτελέσματα του αρχείου input.cme:

lsers\Geurgia\Downl n: PROGRAM n: ID n: NEWLINE n: NEWLINE	loads\opxec\vin_flex_bi Line: 1 Line: 1 Line: 2 Line: 3	ison-latest\nyParser>.\a input Text: PROGRAM Text: input Text: Text:	doken: L_PAR Ioken: STRING 1" Ioken: R_PAR Ioken: R_PAR Ioken: QH Ioken: NEVLINE	Line: 9 Line: 9 Line: 9 Line: 9 Line: 10	Text: {     Text: "The two numbers .     Text: }     Text: ;     Text: ;	OKEN: NEWLINE	Line: 41 Line: 41 Line: 41 Line: 42 Line: 42	Text: > Text: > Text: > Text: Text:
en: COMMENT	Line: 3	Text: %function/if test	Token: ENDIF Token: NEVLINE	Line: 19 Line: 11	Text: ENDIF Text:	oken: PRINT oken: L_PAR oken: STRING oken: R_PAR oken: QM oken: NEWLINE	Line: 42 Line: 42 Line: 42 Line: 42 Line: 42 Line: 43	<pre>lext: PRINT lext: ( lext: "Saturday" lext: ) lext: ; lext: ;</pre>
n: FUNCTION n: ID n: L_PAR n: ID n: ID n: ID n: COMMA n: ID n: ID n: ID	Line: 3 Line: 3 Line: 3 Line: 3 Line: 3 Line: 3 Line: 3 Line: 3 Line: 4	Text: INTEGER Text: x1 Text: Text: INTEGER	Token: RETURN Token: ID Token: QM Token: NEVLINE Token: END_FUNCTION Token: NEVLINE	Line: 11 Line: 11 Line: 12 Line: 12 Line: 12	Text: RETURN Text: x1 Text: ; Text: Text:	oken: BREAK oken: QM Igoken: NEWLINE	Line: 43 Line: 43 Line: 44	Text: BREAK Text:; Text:
n: NEWLINE	Line: 3 Line: 3 Line: 4 Line: 4	Text: x2 Text: > Text: Text:	Token: COMMENT	Line: 13	Text: %function/while to	oken: CASE oken: L_PAR esoken: LT oken: INTEGER	Line: 44 Line: 44 Line: 44 Line: 44 Line: 44 Line: 44 Line: 45	Text: CASE Text: ( Text: { Text: 17 Text: 17 Text: } Text:
n: IF n: L_PAR n: LT n: LT n: ID n: ID n: THEN n: THEN n: NEWLINE	Line: 4 Line: 4 Line: 4 Line: 4 Line: 4 Line: 4 Line: 4 Line: 5	Text: IF Text: ( Text: x1 Text: x Text: ( Text: x2 Text: ) Text: Description Text: THEN	Token: FUNCTION Token: ID Token: L_PAR Token: ID Token: ID Token: ID	Line: 13 Line: 13 Line: 13 Line: 13 Line: 13	Text: doStuff Text: doStuff Text: ( Text: INTEGER Text: var1	oken: GI oken: R_PAR oken: NEWLINE	Line: 44 Line: 44 Line: 45	Taxt: PRINT
- DDTAIT	Line: 4 Line: 5 Line: 5 Line: 5		Token: FUNCTION Token: ID Token: L. PAR Token: ID Token: ID Token: COMM Token: ID Token: ID Token: ID Token: R. PAR Token: NEVLINE	Line: 13 Line: 13 Line: 13 Line: 13 Line: 13 Line: 13 Line: 13 Line: 13 Line: 13 Line: 14	Text: FUNCTION Text: doStuff Text: doStuff Text: UNIGER Text: UNIGER Text: INTEGER Text: INTEGER Text: Var2 Text: Var2 Text: Var2 Text: Var2	oken: PRINT oken: L_PAR oken: STRING oken: R_PAR oken: QM oken: MEWLINE	Line: 45 Line: 45 Line: 45 Line: 45 Line: 45 Line: 46	lext: ( lext: "Sunday" lext: ) lext: ; lext:
L PAR STRING L BRACK COMMA I ID R BRACK	Line: 5	lext PRINT Text ( Text [ Text [ Text x1 Text x1 Text x1 Text x1 Text x1	Token: WHILE Token: L_PAR Token: ID Token: LI	Line: 14 Line: 14 Line: 14 Line: 14	Text: WHILE Text: < Text: var1 Text: <	oken: BREAK oken: QM oken: NEWLINE	Line: 46 Line: 46	<pre>Text: BREAK Text: ; Text:</pre>
NEWLINE	Line: 6	Text:	Token: WHILE Token: L.PAR Token: ID Token: LT Token: INTEGER Token: RND_OP Token: INTEGER TOKEN: RNB_OP TOKEN: RNB_OP TOKEN: RNB_OP TOKEN: RNB_OP	Line: 14 Line: 15	Toxt: WHILE Toxt: VAII Toxt: VAII Text: < Toxt: 10 Toxt: 10 Toxt: 10 Toxt: 0AI Toxt: 20 Toxt: 28 Toxt: 28 Toxt: 28 Toxt: 10 Toxt: 28 Toxt: 10 Toxt:	oken: ENDSVITCH oken: NEWLINE oken: RETURN oken: ID oken: QM oken: MEWLINE	Line: 47 Line: 48 Line: 48 Line: 48 Line: 48 Line: 49	Text: RETURN Text: day Text: ; Text:
: ELSEIF : L_PAR : ID : LT : ID : R_PAR : NEWLINE	Line: 6 Line: 6 Line: 6 Line: 6 Line: 6 Line: 7	Text: ELSEIF Text: ( Text: xi Text: < Text: x2 Text: x2 Text: > Text: >	Token: NEWLINE Token: IF Token: L_PAR Token: ID	Line: 15 Line: 15 Line: 15 Line: 15	Text: Text: IF Text: ( Text: vari	oken: NEWLINE oken: END_FUNCTION oken: NEWLINE	Line: 49 Line: 49 Line: 50	Text: Text: END_FUNCTION Text:
: PRINT : L_PAR : STRING : L_BRACK : COMMA	Line: 7 Line: 7 Line: 7	Text: PRINT Text: ( Text: ""	Ioken: IF Ioken: L.PAR Ioken: ID Ioken: EQ.OP Ioken: ID Ioken: R.PAR Ioken: R.PAR Ioken: NEWLINE	Line: 15 Line: 15 Line: 15 Line: 15 Line: 15 Line: 15 Line: 15 Line: 16	Text: IF Text: { Text: var1 Text: Text: var2 Text: var2 Text: ) Text: IHEN Text: IHEN	oken: STARTMAIN oken: NEWLINE	Line: 50 Line: 51	Text: STARTMAIN Text: UARS
: L_BRHCK : COMMA : ID : R_BRACK : R_PAR : QM : NEWLINE	Line: 7	Text: PRINT Text: (""" Text: ("""" Text: [ Text: / Text: x2 Text: 1 Text: ) Text: ) Text: ) Text: )	Ioken: BREAK Ioken: QM Ioken: NEVLINE	Line: 16 Line: 16 Line: 17	Text: BREAK Text: ; Text:	oken: VARS oken: NEWLINE oken: ID oken: COMMA	Line: 51 Line: 52 Line: 52 Line: 52 Line: 52	Text: UARS Text: Text: CHAR Text: char1 Text:
: ELSE : NEWLINE	Line: 8 Line: 9	Text: ELSE Text:	Token: ENDIF Token: NEWLINE Token: ID Token: ASSIGN	Line: 17 Line: 18 Line: 18 Line: 18	Text: ENDIF Text: Text: var1 Text: -	oken: ID oken: COMMA oken: ID oken: COMMA oken: ID oken: MEWLINE oken: ID	Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53	Text: CHAR Text: char! Text: char! Text: char2 Text: ; Text: ; Text: INTEGER
: PRINT :: L_PAR :: STRING	Line: 9 Line: 9 Line: 9 Line: 9	Text: PRINT Text: ( Text: "The two numbers a Text: >	Token: ID Token: ASSIGN Token: INTEGER Token: QM Langlor NEWLINE Token: NEWLINE	Line: 18 Line: 18 Line: 18 Line: 18 Line: 19 Line: 19	Text: 45 Text: 45 Text: ; Text: Text: yar2	oken: ID oken: COMMA oken: COMMA oken: COMMA oken: COMMA oken: COMMA oken: ID oken: COMMA oken: COMMA oken: COMMA	Line: 53 Line: 54 Line: 54 Line: 55 Lin	Text: Inicorn Text: varr Text: foo Text: fool Text: fool Text: fool
n: R_PAR n: QM n: NEWLINE n: ENDIF n: NEWLINE	Line: 9 Line: 9 Line: 10 Line: 10 Line: 11	Text: ) Text: ; Text: Text: ENDIF Text:	Token: ID Token: GE_OP Token: INTEGER Token: QH Token: MEWLINE	Line: 19 Line: 19 Line: 19 Line: 19 Line: 20	Text: var2 Text: >- Text: 56 Text: ; Text:	oken: COMMA oken: ID oken: COMMA oken: ID oken: COMMA	Line: 53 Line: 53 Line: 53 Line: 53 Line: 53	Text:
: RETURN : ID : QM : NEWLINE	Line: 11 Line: 11 Line: 11 Line: 12	Text: RETURN Text: x1	Token: ENDWHILE Token: NEWLINE Token: RETURN	Line: 28 Line: 21 Line: 21	Text: ENDUHILE Text: Text: RETURN	oken: COMMA oken: IDMMA oken: IDM oken: COMMA oken: COMMA oken: COMMA oken: LBRACK oken: LBRACK oken: LBRACK oken: LBRACK oken: LBRACK oken: RAMA oken: HMEGER oken: OMMA	Line: 53 Line: 53 Line: 53 Line: 53 Line: 53	Text: Foo4 Text: foo4 Text: foo5 Text: foo5 Text: foo5
: END_FUNCTION : NEWLINE	Line: 12 Line: 13	Text: ; Text:  Iext: END_FUNCTI Text:	Oloken: ID Oloken: QM Token: NEVLINE	Line: 21 Line: 21 Line: 21 Line: 22	Text: RETURN Text: var2 Text: ; Text:	oken: L_BRACK oken: INTEGER oken: R_BRACK	Line: 53 Line: 53	Text: Î Text: 100 Text: ]
OAMMENT			Token: END_FUNCTION	Line: 22	Text: END FUNCT	oken: R_BRACK oken: COMMA ICoken: ID	Line: 53 Line: 53 Line: 53	Text: 108 Text: 1 Text: 1 Text: 4
	Line: 13	Text: Xfunction/while te	Token: END_FUNCTION 18 Token: NEWLINE Token: ID		Text: END_FUNCT	Taken: R BROCK		Text: ]
: STARTMAIN : NEVLINE : VARS : NEVLINE	Line: 58 Line: 51 Line: 52	Text: %function/while te Text: STARTMAIN Text: UARS Text:	Token: END_FUNCTION PEULINE  Token: ID Token: ASSIGN	Line: 22 Line: 23	Text: END_FUNCT	Ioken: R_BRACK Ioken: ASSIGN Ioken: INTEGER Ioken: QM Ioken: NEWLINE	Line: 64 Line: 64 Line: 64 Line: 65	Text: day
: STARTMAIN : NEVLINE : VARS : VARS : ID : ID : COHMA : ID : QH	Line: 58 Line: 51 Line: 52	Text: %function/while te Text: STARTMAIN Text: UARS Text:	Token: END FUNCTION  LOKEN: NEULINE  LOKEN: 1D  LOKEN:	Line: 22 Line: 23 Line: 57 Line: 58	Text: END_PUNCT  Text: foo4	Ioken: R_BRACK Ioken: ASSIGN Ioken: INTEGER Ioken: QM Ioken: NEWLINE	Line: 64 Line: 64 Line: 64 Line: 65	iest: day  Foxt: 1  Foxt: 30  Foxt: :  Foxt: -  Foxt: -  Toxt: -  Toxt: pinakes
: STORTMAIN : NEVLINE : UARS : NEVLINE : ID : ID : COMMA : ID	Line: 51 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53	Text: Xfunction/while te Text: SIGRIPAIN Text: SIGRIPAIN Text: CHOR Text: CHOR Text: CHOR Text: Chor Text: Laber Text: Laber Text: Laber Text: Laber Text: Laber Text: Laber	Token: MU.RUKCTION  TOKEN: MEWLINE  TOKEN: SETON  TOKEN: METERS  T	Line: 22 Line: 23 Line: 57 Line: 58	Text: END_PUNCT  Text: foo4	Token: R_BRACK Token: ASSIGN Token: INTEGER	Line: 64 Line: 64 Line: 64 Line: 65	lext: day lext: j Text: j Text: 30 Text: ; Text: vinekas
: STORTHAIN : NEULINE : UARS : NEULINE : ID : ID : ID : COMMA : ID	Line: 51 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53	Text: Xfunction/while te Text: SIGRIPAIN Text: SIGRIPAIN Text: CHOR Text: CHOR Text: CHOR Text: Chor Text: Laber Text: Laber Text: Laber Text: Laber Text: Laber Text: Laber	Loben: NO. PINCTION  TOWN: NOV. IN THE CONTROL OF T	Line: 23 Line: 22  Line: 57 Line: 58 Li	Text: Foot: EMO_FUNCT  Text: feed foot: foot: foot: foot: foot: foot: foo: foo	Johann R. R. RRACK I Johann R. R. S. S. GM I Johann R. S. S. GM I Johann R. R. L. R. R. L. R. R. L. R. L. R. R. L. R. L. R. R. R. L. R. R. R. L. R. R. R. L. R. R. R. R. L. R. R. R. R. L. R. R. R. R. R. R. L. R.	Line: 64 Line: 64 Line: 64 Line: 64 Line: 65	Text:
: STORTMAIN : NEVLINE : UARS : NEVLINE : ID : ID : COMMA : ID	Line: 51 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53	Text: Xfunction/while te Text: SIGRIPAIN Text: SIGRIPAIN Text: CHOR Text: CHOR Text: CHOR Text: Chor Text: Laber Text: Laber Text: Laber Text: Laber Text: Laber Text: Laber	Token: PMD_RINCTION  Token: SEVICE  Token: SEVICE  Token: SEVICE  Token: SEVICE  Token: DPD  Token: OPD  Token: OP	Line: 22 Line: 57 Line: 58 Line: 59 Lin	Text: EMD_FUNCT Text: fee4 feet: fee4 feet: fee3 feet: fee3 feet: fee3 feet: fee3 feet: fee5 feet: fee5 feet: fee5 feet: fee5 feet: fee5 feet: fee5 feet: fee6 fee6 fee6 fee6 fee6 fee6 fee6 fee6	Johann R. R. RRACK I Johann R. R. S. S. GM I Johann R. S. S. GM I Johann R. R. L. R. R. L. R. R. L. R. L. R. R. L. R. L. R. R. R. L. R. R. R. L. R. R. R. L. R. R. R. R. L. R. R. R. R. L. R. R. R. R. R. R. L. R.	Line: 64 Line: 64 Line: 64 Line: 64 Line: 65	Text:
: STORTMAIN : NEVLINE : UARS : NEVLINE : ID : ID : COMMA : ID	Line: 51 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53	Text: Xfunction/while te Text: SIGRIPAIN Text: SIGRIPAIN Text: CHOR Text: CHOR Text: CHOR Text: Chor Text: Laber Text: Laber Text: Laber Text: Laber Text: Laber Text: Laber	Token: HD PRINCTION  TOKEN: STATE TOKEN  TOKEN: TOK	Line: 22 Line: 57 Line: 58 Line: 59 Lin	Text: EMD_FUNCT Text: fee4 feet: fee4 feet: fee3 feet: fee3 feet: fee3 feet: fee3 feet: fee5 feet: fee5 feet: fee5 feet: fee5 feet: fee5 feet: fee5 feet: fee6 fee6 fee6 fee6 fee6 fee6 fee6 fee6	I Oken: R. BRÄCK I Oken: R. SS. I GN I Oken: R. SS. I GN I Oken: MT BEGER I Oken: I MT BEGER I Oken: MT BEGER I Oken: I MT BEGER I Oken: I MT BEGER I Oken: R. BRÄCK I Oken: R. BRÄCK I Oken: R. BRÄCK I Oken: MT BEGER I Oken: R. BRÄCK I Oken: MT BEGER I Oken: R. BRÄCK I Oken: R. BRÄLLI NE	Line: 64 Line: 64 Line: 64 Line: 65 Line: 66 Line: 67	Text:
E TARTHALN  FENULINE  FAULINE  1 MARS  1 DI  1 DI  1 DI  1 OPPRI  1 DI  1 OPPRI  1 DI  1 D	Line: 58 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53 Line: 55 Line: 56 Lin	Text: Zfunction/while to  Text: VARE  Text: VARE  Text: VARE  Text: VARE  Text: Charl  Text: Charl  Text: Charl  Text: Text: Text  Text  Text: Text	Token BHD_RINCTION  TOKEN BHY	Line: 22  Line: 57  Line: 58  Line: 59  Line: 68  Line:	Text: EMO_FUNCT Text: feed feet feed feet destuff feet feed feet	JOHENS R. BRACK JOHEN R. SSIGN JOHEN R. SSIGN JOHEN R. MERCER JOHEN L. BRACK JOHEN R. STIGN JOHEN JOHEN JOHEN JOHEN JOHEN JOHEN JOHEN JOH	Line: 64 Line: 64 Line: 64 Line: 65 Line: 66 Line: 67	Text:
ETRIPPHIN PROLINE PROLINE 1 DATE 1 DA	Line: 58 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53 Line: 54 Line: 54 Line: 54 Line: 55 Line: 56 Lin	Text: Zfunction/while to Text: VARE Text: VARE Text: VARE Text: VARE Text: Char Text: Char Text: Char Text: Char Text: Text: Text Text: Calculations Text: Calculations	Token: BHD_RINCTION  TOKEN: SET ON	Line: 22 Line: 57 Line: 58 Line: 59 Lin	Text: EMO_FUNCT Text: [se4 Text: destuff Text: destuff Text: destuff Text: feel Text: fe	IO KERT. R. BRACK IO KERT. R. SES IGN IO KERT.	Line: 64 Line: 64 Line: 64 Line: 65 Line: 66 Line: 67 Line: 68 Line: 68	Text:
ETARINAIN NEULINE 1 UARE 1 UAR	Line: 58 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53 Line: 54 Line: 55 Line: 56 Lin	Text: zfunction/while te  Text: VORE: Text: VORE: Text: VORE: Text: Char: Text: Char: Text: Char: Text: Char: Text: Char: Text: Char: Text: Text	Token: BHD_RINCTION  TOKEN: SET ON	Line: 22  Line: 57  Line: 58  Line: 59  Line: 68  Line:	Text: EMO_FUNCT Text: [se4 Text: destuff Text: destuff Text: destuff Text: feel Text: fe	IO KERT. R. BRACK IO KERT. R. S. S. G. G. H. IO KERT. R. S. S. G. G. H. IO KERT. R. S. G.	Line: 64 Line: 64 Line: 64 Line: 65 Line: 66 Line: 67 Line: 68	Text:
ETARTHAIN REVLINE USINE	Line: 58 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53 Line: 54 Line: 55 Line: 56 Line: 57 Line: 58 Lin	Text: Zfunction/while te  Text: WARE  Text: WARE  Text: WARE  Text: WARE  Text: WARE  Text: Chard  Text: Chard  Text: Chard  Text: Chard  Text: Chard  Text: Text  Text: X calculations  Text: X calculation	Johan BHO, PHINCTION  JOHAN BENGLINE  JOHAN JOHAN BENGLINE  JOHAN BENGLINE  JOHAN JOHAN BENGLINE	Line: 22  Line: 27  Line: 57  Line: 58  Line: 59  Line: 68  Line:	Text: EMO_FUNCT  Text: feed  T	IO KERT R. BRACK IO KERT R. S. S. G. G. H. IO KERT R. S. S. G. G. H. IO KERT R. S. G.	Line: 64 Line: 64 Line: 64 Line: 65 Line: 65 Line: 65 Line: 65 Line: 65 Line: 65 Line: 66 Line: 67 Line: 67 Line: 67 Line: 67 Line: 67 Line: 68	Text:
ETARTHAIN NEULINE 1 UARS 1 UAR	Line: 58 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53 Line: 54 Line: 55 Line: 56 Line: 57 Line: 58 Lin	Text: Zfunction/while te  Text: WARE  Text: WARE  Text: WARE  Text: WARE  Text: WARE  Text: Chard  Text: Chard  Text: Chard  Text: Chard  Text: Chard  Text: Text  Text: X calculations  Text: X calculation	Johan BHO, PHINCTION  JOHAN BENGLINE  JOHAN JOHAN BENGLINE  JOHAN BENGLINE  JOHAN JOHAN BENGLINE	Line: 22  Line: 27  Line: 57  Line: 58  Line: 59  Line: 68  Line:	Text: EMO_FUNCT  Text: feed  T	IO KERT R. BRACK IO KERT R. S. S. G. G. H. IO KERT R. S. S. G. G. H. IO KERT R. S. G.	Line: 64 Line: 64 Line: 64 Line: 65 Line: 65 Line: 65 Line: 65 Line: 65 Line: 65 Line: 66 Line: 67 Line: 67 Line: 67 Line: 67 Line: 67 Line: 68	Text:
ETARTHAIN NEULINE 1 UARS 1 UAR	Line: 58 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53 Line: 54 Line: 55 Line: 56 Line: 57 Line: 58 Lin	Text: Zfunction/while te  Text: WARE  Text: WARE  Text: WARE  Text: WARE  Text: WARE  Text: Chard  Text: Chard  Text: Chard  Text: Chard  Text: Chard  Text: Text  Text: X calculations  Text: X calculation	Toben: DOP, PINCTION  TOBEN: SEL CALL  T	Line: 22  Line: 57  Line: 58  Line: 59  Line: 50  Line: 60  Line: 61  Line: 61  Line: 61  Line: 62  Line: 62  Line: 62  Line: 62  Line: 62  Line: 62  Line: 63  Line: 64  Line: 63  Line: 64  Line:	Text: EMO_FUNCT  Text: feed  feet feed  feet feed  feet feet  feet	IONEM: R.BRACK IONEM: ASSIGN IONEM: ASSIGN IONEM: INTEGER IONEM: RASSIGN IONEM: RASSIGN IONEM: RASSIGN IONEM: RASSIGN IONEM: RASSIGN IONEM: INTEGER IONEM: I	Line: 64 Line: 64 Line: 64 Line: 65 Line: 66 Line: 67 Line: 68 Line: 69 Lin	Text:     Text:     Text:     Text:     Text:   30     Text:     Text:   yinakas     Text:     Text:   yinakas     Text:   y
ETARTHAIN HENLINE 1001 1001 1001 1001 1001 1001 1001 10	Line: 58 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53 Line: 54 Line: 55 Line: 57 Lin	Text: *#Innetion*/shile te  Text: **Innetion*/shile te  Text: **OBRE**  Text: **OBRE**  Text: **CAND**  Text: **CAND**  Text: **CAND**  Text: **CAND**  Text: **CAND**  Text: **Too'  Te	Johan BHO, RINGTION  JOHAN BHO	Line: 22  Line: 27  Line: 57  Line: 58  Line: 59  Line: 60  Line:	Text: END_FUNCT  Text: feed  T	IONEM: R.BRACK IONEM: ASSIGN IONEM: ASSIGN IONEM: INTEGER IONEM: RASSIGN IONEM: RASSIGN IONEM: RASSIGN IONEM: RASSIGN IONEM: RASSIGN IONEM: INTEGER IONEM: I	Line: 64 Line: 64 Line: 64 Line: 65 Line: 66 Line: 67 Line: 68 Line: 69 Lin	Text:
I COMMENT  I ETART PALIN  I ETART PALIN  I ENULINE  I PULINE  I PU	Line: 58 Line: 51 Line: 52 Line: 52 Line: 52 Line: 52 Line: 52 Line: 53 Line: 54 Line: 55 Line: 56 Line: 57 Line: 58 Lin	Text: Zfunction/while te  Text: WARE  Text: WARE  Text: WARE  Text: WARE  Text: WARE  Text: Chard  Text: Chard  Text: Chard  Text: Chard  Text: Chard  Text: Text  Text: X calculations  Text: X calculation	Johan BHO, PHINCTION  JOHAN BENGLINE  JOHAN JOHAN BENGLINE  JOHAN BENGLINE  JOHAN JOHAN BENGLINE	Line: 22  Line: 27  Line: 57  Line: 58  Line: 59  Line: 68  Line:	Text: EMO_FUNCT  Text: feed  T	IO KERT R. BRACK IO KERT R. S. S. G. G. H. IO KERT R. S. S. G. G. H. IO KERT R. S. G.	Line: 64 Line: 64 Line: 64 Line: 65 Line: 65 Line: 65 Line: 65 Line: 65 Line: 65 Line: 66 Line: 67 Line: 67 Line: 67 Line: 67 Line: 67 Line: 68	Text:     Text:     Text:     Text:     Text:   30     Text:     Text:   yinakas     Text:     Text:   yinakas     Text:   y

Token: ASSIGN Token: INTEGER Token: QM Token: NEWLINE	Line: 69	Text: =
Token: INIEGER	Line: 69 Line: 69	Text: 80 Text: ;
Token: NEWLINE	Line: 70	Text:
Token: ID	Line: 70	Text: pinakas
Token: L_BRACK	Line: 70 Line: 70	Text: [ Text: 90
Token: INIEGER	Line: 70 Line: 70	Text: ]
Token: INTEGER Token: R_BRACK Token: ASSIGN	Line: 70	Text: =
Token: INTEGER Token: QM Token: NEWLINE	Line: 70 Line: 70	Text: 90
Token: QM	Line: 70 Line: 71	Text: ; Text:
Token: NEWLINE	Line: 71	Text:
Token: ID	Line: 71	Text: pinakas
Token: L_BRACK Token: INTEGER Token: R_BRACK	Line: 71 Line: 71 Line: 71 Line: 71 Line: 71	Text: [
Token: INTEGER	Line: 71	Text: Î Text: 100
Token: R_BRACK Token: ASSIGN	Line: 71	Text: 1 Text: =
loken: HSSIGN	Line: 71 Line: 71	lext: =
Token: INTEGER Token: QM	Line: 71	Text: 100 Text: ;
Token: NEWLINE	Line: 71 Line: 71 Line: 71 Line: 72	Text:
Token: FOR	Line: 72	Text: FOR
Token: ID	Line: 72	Text: varr
	Line: 72	Text: :
Token: ASSIGN	Line: 72 Line: 72	Text =
Token: INTEGER Token: TO	Line: 72	Text: 10 Text: TO Text: 100
Token: TO Token: INTEGER	Line: 72 Line: 72	Text: 10
Token: INTEGER	Line: 72 Line: 72	Text: STEP
Token: STEP Token: INTEGER	Line: 72 Line: 72 Line: 72 Line: 73	Text: 10
Token: NEWLINE	Line: 73	Text: 10 Text:
T-1 DDINT	14 12	T DDINT
Token: PRINT Token: L_PAR Token: STRING	Line: 73 Line: 73 Line: 73 Line: 73 Line: 73 Line: 73	Text: PRINT
Token: STRING	Line: 73	Text: ( Text: ""
Token: L BRACK	Line: 73	Text: [ Text: ,
Token: COMMA Token: ID	Line: 73	Text:
Token: ID	Line: 73 Line: 73	Text: pinakas Text: [
Token: L_BRACK Token: ID	Line: 73 Line: 73	Text: L Text: varr
Token: ID Token: R_BRACK Token: R_BRACK	Line: 73	Text: varr Text: l
Token: R_BRACK	Line: 73	Text: 1
Token: R_PAR	Line: 23	Text: >
Token: QM Token: NEWLINE	Line: 73 Line: 74	Text: ; Text:
TOKEH- NEWLINE	Line. 74	lext:
Token: ENDFOR	Line: 74 Line: 75	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 Line: 76	Text: swissFiss Text: (
Token: L_PAR	Line: 76 Line: 76	Text: (
Token: IV Token: R PAR	Line: 76 Line: 76	Text: day Text: >
Token: ID Token: R_PAR Token: QM	Line: 76	Tevt: :
Token: NEWLINE	Line: 77	Text:
Token: COMMENT	Line: 77	
TOKEH - COMMENT	nine: 77	Text: %end of program.:>
Token: ENDMAIN	Line: 77	Text: ENDMAIN
Token: NEWLINE	Line: 78	Text:
Parsing finished	succesfullu!	
- I o zna z znasneu	I wood arry.	
LUAME	LINTEGER	
: NAME	: INTEGER :	
sub	<del> </del> −20	
l plus	¦ 110 ¦	
¦ div ! mul	: 6 : : 200 :	
: mul	1 200 1	
C:\Users\Gewrgia	\Downloads\opxec\win_f	lex_bison-latest\myParser>