Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Project 2020-21

Κατερίνα Μητροπούλου (1067409)
Μάριος Στεφανίδης (1067458)
Δημήτρης Βλαχογιάννης (1067371)

Περιεχόμενα

BNF γραμματική

Βασικοί Ορισμοί

- <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- <char> ::= a | ... | z | A | ... | Z
- <integer> ::= [-] (<digit> | <digit> <integer>)
- <string_symbol> ::= "(" | ")" | "[" | "]" | "{" | "}" | ? | : | ; | - | , | !
- <variable> ::= <type_specifier> <var_name>["["<positive_integer>"]"]

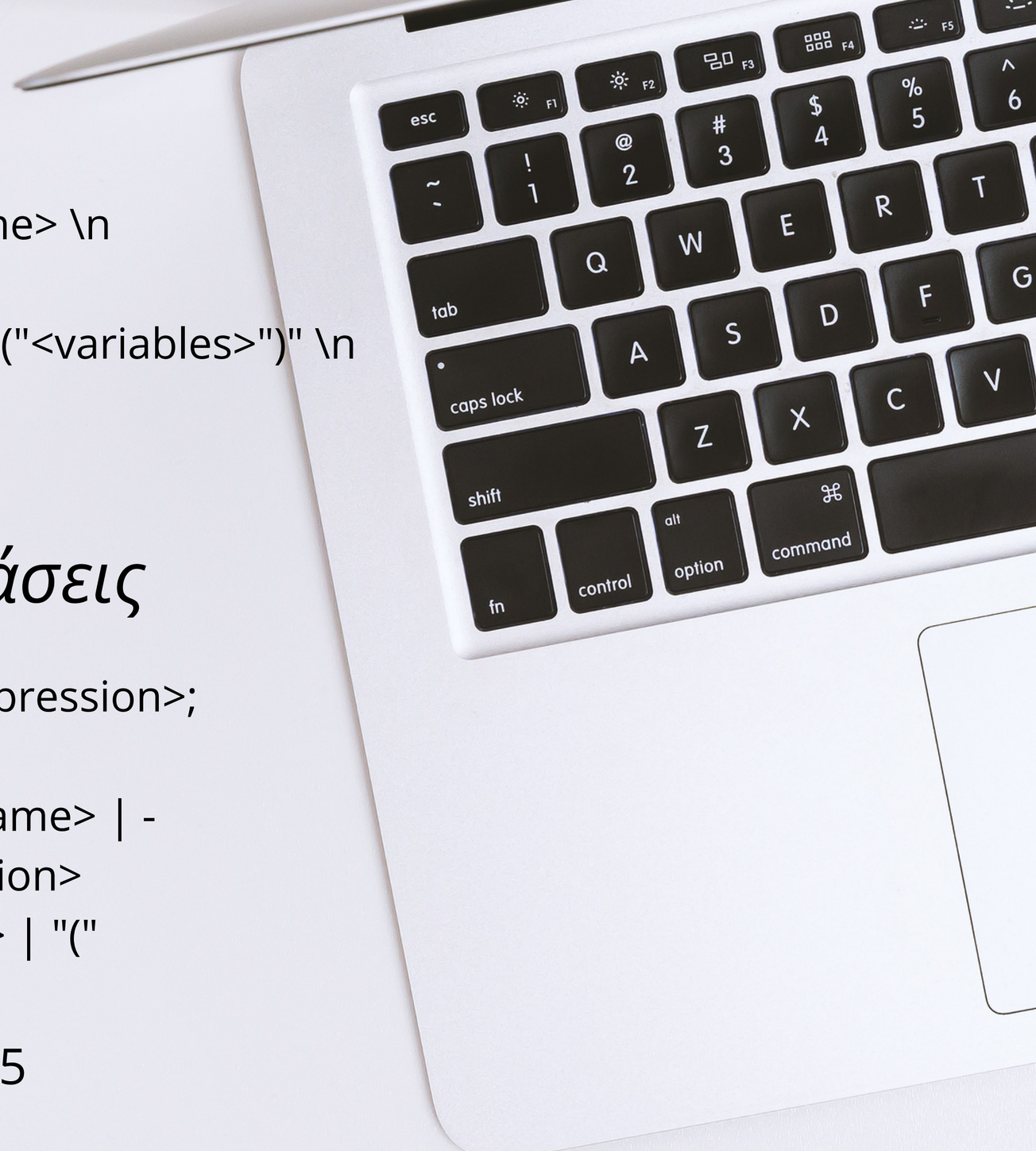
- <type_specifier> ::= INT | CHAR
- <positive_integer> ::= <digit> | <digit> <positive_integer>
- <var_name> ::= ((_|<char>) [<var_tail>])
- <var_tail> ::= <char> | <digit> | <digit> <var_tail> | <char> <var_tail> | _<var_tail>
- <variables> ::= <variable> | <variable>, <variables>
- <string> ::= <char> | <digit> | <char> <string> | <digit> <string> | <string_symbol> <string>
- <arithmetic_operator> ::= + | - | ^ | * | / | > | < | == | !=
- <logic_operator> ::= AND | OR

Δηλώσεις

- <program_declaration> ::= PROGRAM <var_name> \n
- <function_declaration> ::= FUNCTION <string> "("<variables>")" \n
- <variable_declaration> ::= VARS <variables>;

Αριθμητικές και Λογικές εκφράσεις

- <assignment> ::= <var_name> = <arithmetic_expression>;
- <arithmetic_expression> ::= <integer> | <var_name> | -<arithmetic_expression> | <arithmetic_expression> <arithmetic_operator> <arithmetic_expression> | "(" <arithmetic_expression>")"



- <logic_expression> ::= <integer> | <var_name> | <logic_expression> <logic_operator> <logic_expression> | <arithmetic_expression> | "(" <logic_expression> ")'

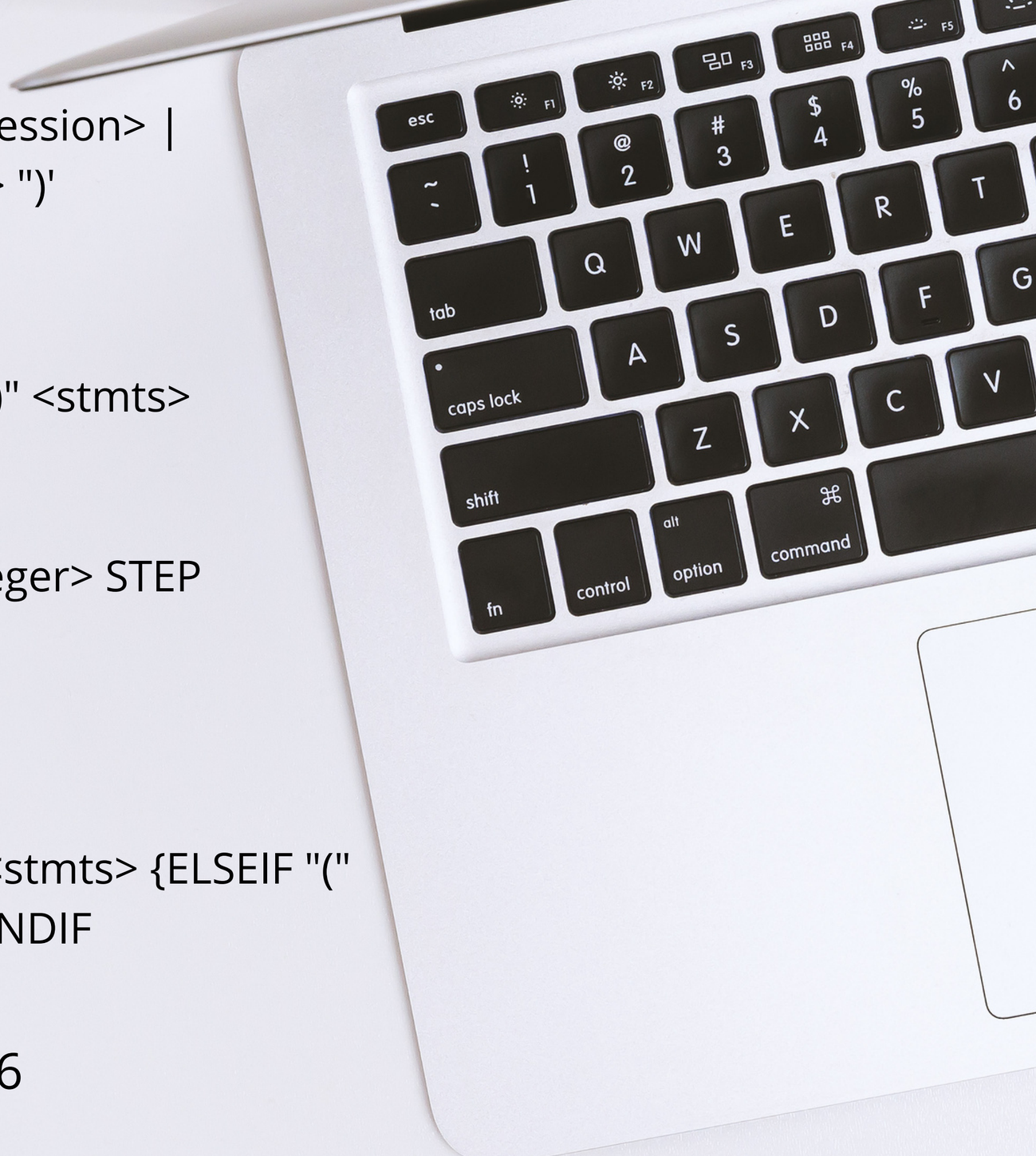
Εντολές Επανάληψης

- <while_stmt> ::= WHILE "(" <logic_expression> ")" <stmts> "ENDWHILE"

- <for_stmt> ::= FOR counter:= <integer> TO <integer> STEP <integer> <stmts> ENDFOR

Εντολές Ελέγχου

- <if_stmt> ::= IF "(" <logic_expression> ")" THEN <stmts> {ELSEIF "(" <logic_expression ")" <stmts>} {ELSE <stmts>} ENDIF



- <switch_stmt> ::= SWITCH "(" <logic_expression> ")" CASE "(" <logic_expression> ")": <stmts> {CASE "(" <logic_expression> ")": <stmts>} [DEFAULT: <stmts>] ENDSWITCH

Εντολή Εκτύπωσης στην οθόνη

- <print_stmt> ::= PRINT "(" \"<string>\" ["[", <var_name> "]"] ")";

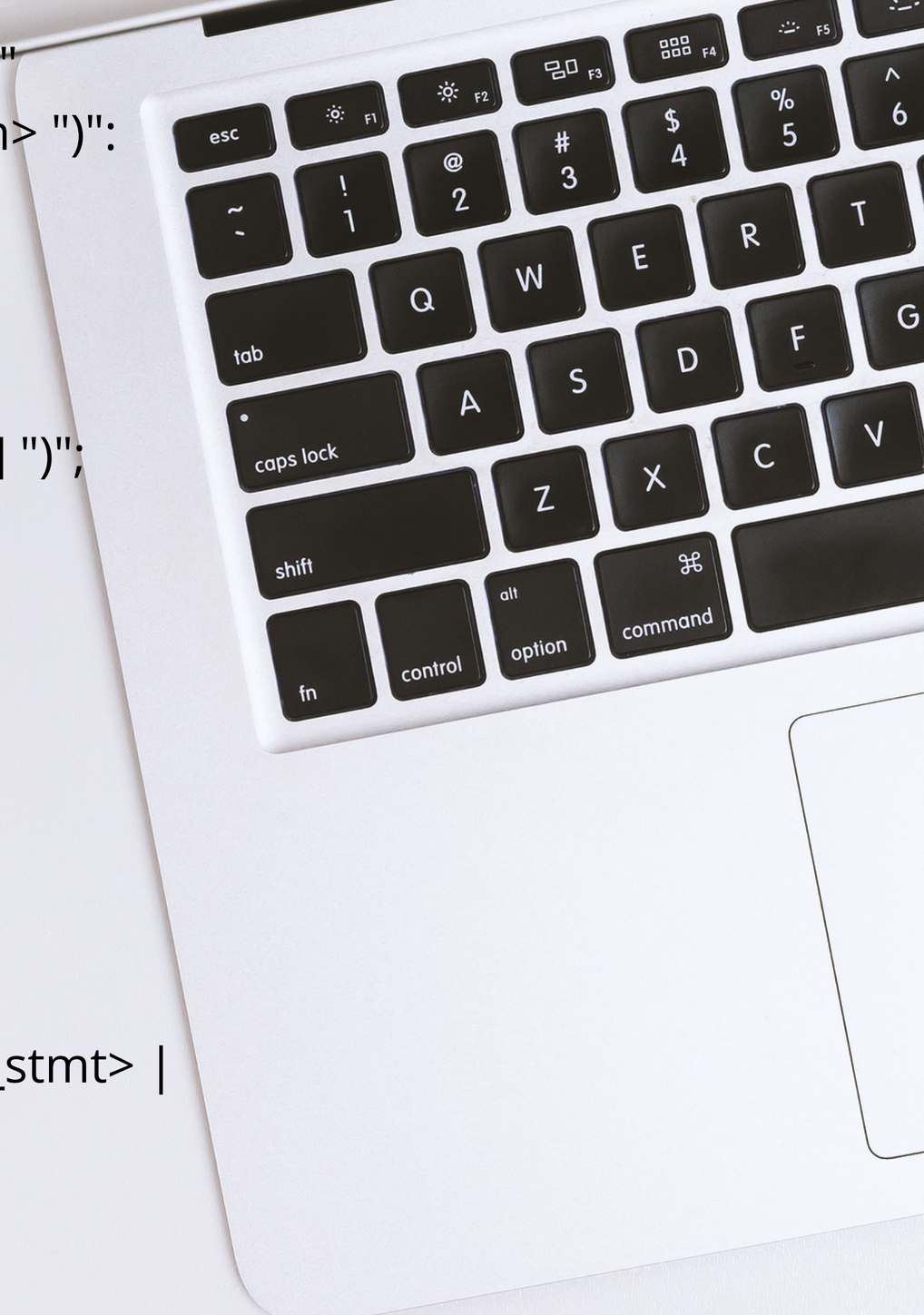
Σχόλια μιας γραμμής

- <comment_stmt> ::= % <string>

Εξίσου σημαντικά στοιχεία

- <stmts> ::= <stmt> | <stmt> <stmts>

- <stmt> ::= <assignment> | <while_stmt> | <for_stmt> | <if_stmt> | <switch_stmt> | <print_stmt> <comment_stmt> | BREAK;



- <function> ::= <function_declaration> <variable_declaration> <stmts> RETURN <arithmetic_expression>; END_FUNCTION

- <main> ::= STARTMAIN <variable_declaration> <stmts> ENDMAIN

- <program_body> ::= <program_declaration> {<function>} <main>

Υποδείξεις:

1. {...} -> 0 ή περισσότερες εμφανίσεις
2. [...] -> προαιρετικό
3. (...) -> ομαδοποίηση
4. Σύμβολα που χρησιμοποιούνται από την γραμματική BNF και χρειάστηκε να συμπεριληφθούν στην παραγωγή των κανόνων, συμβολίζονται μέσα σε εισαγωγικά.
5. Οι λέξεις με κεφαλάια είναι δεσμευμένες

Λεκτικός Αναλυτής - Flex

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

int lineno = 1;
void t_print();
extern void yyerror(char *message);
%}

%option noyywrap
%option yylineno

%x STATE_ONELINE_COMMENT
%x STATE_MULTILINE_COMMENT

letter      [a-zA-Z]|[0-9]
digit       [0-9]
pos_int     {digit}+
ID          _*[a-zA-Z_][0-9a-zA-Z_]*

%%
```

```
"PROGRAM"           {t_print(); return T_PROGRAM;}
"FUNCTION"          {t_print(); return T_FUNCTION;}
"VARS"              {t_print(); return T_VARS;}
"INT"               {t_print(); return T_INT;}
"CHAR"              {t_print(); return T_CHAR;}
"WHILE"             {t_print(); return T_WHILE;}
"ENDWHILE"          {t_print(); return T_ENDWHILE;}
"FOR"               {t_print(); return T_FOR;}
"counter"           {t_print(); return T_COUNTER;}
"TO"                {t_print(); return T_TO;}
"STEP"              {t_print(); return T_STEP;}
"ENDFOR"            {t_print(); return T_ENDFOR;}
"IF"                {t_print(); return T_IF;}
"THEN"              {t_print(); return T_THEN;}
"ELSEIF"            {t_print(); return T_ELSEIF;}
"ELSE"              {t_print(); return T_ELSE;}
"ENDIF"             {t_print(); return T_ENDIF;}
"SWITCH"            {t_print(); return T_SWITCH;}
"CASE"              {t_print(); return T_CASE;}
"DEFAULT"           {t_print(); return T_DEFAULT;}
"ENDSWITCH"         {t_print(); return T_ENDSWITCH;}
"PRINT"             {t_print(); return T_PRINT;}
"BREAK"             {t_print(); return T_BREAK;}
```

Λεκτικός Αναλυτής - Flex

```
"RETURN"                {t_print(); return T_RETURN;}
"END_FUNCTION"          {t_print(); return T_ENDFUNCTION;}
"STARTMAIN"             {t_print(); return T_STARTMAIN;}
"ENDMAIN"               {t_print(); return T_ENDMAIN;}
"STRUCT"                {t_print(); return T_STRUCT;}
"ENDSTRUCT"             {t_print(); return T_ENDSTRUCT;}
"TYPEDEF"               {t_print(); return T_TYPEDEF;}

\"(\\.|[^\\"\n])*\"     {t_print(); return T_STRING;}
\'(\\.|[^\\"\n])\'      {t_print(); return T_CHARACT;}

"="             {t_print(); return T_ASSIGN;}
";"             {t_print(); return T_SEMI;}
"("             {t_print(); return T_LPAREN;}
")"             {t_print(); return T_RPAREN;}
":"             {t_print(); return T_COLON;}
"["             {t_print(); return T_LBRACK;}
"]"             {t_print(); return T_RBRACK;}
"+"|"-"         {t_print(); return T_ADDOP;}
"{"             {t_print(); return T_LBRACE;}
"}"             {t_print(); return T_RBRACE;}
","             {t_print(); return T_COMMA;}
```

```
"^"|"*"|"/"                 {t_print(); return T_MULOP;}
"=="|"!="                   {t_print(); return T_EQOP;}
">"|"<"                     {t_print(); return T_RELOP;}
"AND"|"OR"                  {t_print(); return T_LOGOP;}
\n|\r\n|\r                  {printf("\n"); lineno++;}
"\t"                        {printf("\t");}
" "                         {printf(" ");}

{pos_int}                   {t_print(); return T_NUMBER;}
{ID}        {t_print(); yylval.strval = strdup(yytext); return T_ID;}

"%"                         {BEGIN(STATE_ONELINE_COMMENT);}
<STATE_ONELINE_COMMENT>.                                  {}
<STATE_ONELINE_COMMENT>\n|\r\n|\r   {BEGIN(INITIAL); lineno++;}

"/*"                        {BEGIN(STATE_MULTILINE_COMMENT);}
<STATE_MULTILINE_COMMENT>\n|\r\n|\r                {lineno++;}
<STATE_MULTILINE_COMMENT>.                                {}
<STATE_MULTILINE_COMMENT>"*/"                {BEGIN(INITIAL);}

.                           {yyerror("Wrong token");}
<<EOF>>                     {t_print(); return T_EOF;}

%%
```

```
void t_print()
{
    printf("%s", yytext);
 }
```

Λεκτικός Αναλυτής - Flex

Παρατηρήσεις:

- Ορίζουμε τα tokens, από τα οποία αποτελείται η C-like γλώσσα, και θέλουμε να αναγνωρίζει ο λεκτικός αναλυτής, ώστε να τα επιστρέφει στον αντίστοιχο συνακτικό.
- Προκειμένου να υλοποιήσουμε one-line και multi-line comments εκμεταλευόμαστε την λειτουργία state του flex.
  Σχόλια μιας γραμμής: Όταν διαβάζει τον χαρακτήρα % μπαίνει στο αντίστοιχο state (STATE_ONELINE_COMMENT), όπου οποιονδήποτε χαρακτήρα κι αν διαβάσει, δεν εκτελεί κώδικα (ουσιαστικά τον "αγνοεί"). Από αυτό το state βγαίνει την στιγμή που διαβάσει τον χαρατήρα αλλαγή γραμμής \n.
  Σχόλια πολλαπλών γραμμών: Λειτουργούν όπως και τα σχόλια μιας γραμμής, μόνο που ο flex μπαίνει στο αντίστοιχο state (STATE_MULTILINE_COMMENT), όταν διαβάζει τον χαρακτήρα /* και εξέρχεται από αυτόν (INITIAL), όταν συναντήσει τον χαρακτήρα */.
- Θεωρούμε ότι κάθε string (T_STRING) περιέχεται μέσα σε εισαγωγικά (" "), το εσωτερικό των οποίων μπορεί να αποτελείται από οτιδήποτε εκτός από το σύμβολο ".
- Αντίστοιχα, κάθε char (T_CHARACT) περιέχεται μέσα σε ' ' και μπορεί να αποτελείται από έναν μόνο χαρακτήρα είτε αριθμό, είτε γράμμα της αλφαβήτου, είτε σύμβολο, εκτός όμως από το ".
- Επιπλέον, αρχικοποιούμε μια μεταβλητή lineno, η οποία κάθε φορά που συναντά το σύμβολο αλλαγής γραμμής αυξάνεται κατά ένα. Κατά αυτό τον τρόπο, μπορούμε να γνωρίζουμε εύκολα κάθε φορά που εμφανίζεται κάποιο error σε ποια γραμμή βρίσκεται αυτό.

Συντακτικός Αναλυτής - Bison

```
"%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include "symbol_table.h"

extern int lineno;
extern int yylex();
extern char *yytext;
extern FILE *yyin;
int cline = 0;

void yyerror(const char *message);

void putvar(char *var_name)
{
  variable *s;
  s = search(var_name);

  if (s == 0)
  {
      s = insert(var_name);
  }
  else
  {
      printf("\t%s is already defined.", var_name);
      exit(0);
  }
}
```

```
void checkvar(char *var_name)
{
   if (search(var_name) == 0)
   {
      printf("\t%s is an undeclared identifier.", var_name);
      exit(0);
   }
}

void checkline()
{
   if(cline==lineno)
   {
      yyerror("new line is required");
      exit(0);
   }
}

%}

%error-verbose

%union {
   int intval;
   char* strval;
}

%token T_PROGRAM      "PROGRAM"
%token T_FUNCTION     "FUNCTION"
%token T_VARS         "VARS"
```

Συντακτικός Αναλυτής - Bison

```
%token T_ASSIGN "="
%token T_SEMI ";"
%token T_LPAREN "("
%token T_RPAREN ")"
%token T_WHILE "WHILE"
%token T_ENDWHILE "ENDWHILE"
%token T_FOR "FOR"
%token T_COUNTER "counter"
%token T_TO            "TO"
%token T_STEP          "STEP"
%token T_ENDFOR        "ENDFOR"
%token T_IF            "IF"
%token T_THEN          "THEN"
%token T_ELSEIF        "ELSEIF"
%token T_ELSE          "ELSE"
%token T_ENDIF         "ENDIF"
%token T_SWITCH        "SWITCH"
%token T_CASE          "CASE"
%token T_COLON         ":"
%token T_DEFAULT       "DEFAULT"
%token T_ENDSWITCH     "ENDSWITCH"
%token T_PRINT         "PRINT"
%token T_LBRACK        "["
%token T_RBRACK        "]"
%token T_BREAK         "BREAK"
%token T_RETURN        "RETURN"
%token T_ENDFUNCTION   "ENDFUNCTION"
%token T_STARTMAIN     "STARTMAIN"
%token T_ENDMAIN       "ENDMAIN"
%token T_ADDOP         "+|-"
%token T_LBRACE        "{"
```

```
%token T_RBRACE "}"
%token T_COMMA ","
%token T_MULOP "^|*|/"
%token T_EQOP "==|!="
%token T_RELOP "<|>"
%token T_LOGOP "AND|OR"
%token T_EOF 0
%token T_CHAR "CHAR"
%token T_INT "INT"
%token <strval>T_ID "identifier"
%token <intval>T_NUMBER "positive integer"
%token <strval>T_STRING "string"
%token <strval>T_CHARACT "character"
%token T_STRUCT "STRUCT"
%token T_ENDSTRUCT "ENDSTURCT"
%token T_TYPEDEF "TYPEDEF"

%left  T_COMMA
%right T_ASSIGN
%left  T_LOGOP
%left  T_EQOP
%left  T_RELOP
%left  T_ADDOP
%left  T_MULOP
%left  T_LPAREN T_RPAREN T_LBRACK T_RBRACK

%start program_body

%%
```

Συντακτικός Αναλυτής - Bison

program_body: program_declaration typedef_decl function main;

typedef_decl: T_STRUCT {checkline(); cline=lineno;} T_ID T_VARS {checkline();} vars
T_ENDSTRUCT
| T_TYPEDEF {checkline(); cline=lineno;} T_STRUCT T_ID T_VARS {checkline();} vars T_ID
T_ENDSTR
;

vars: variable T_SEMI
| variable T_SEMI vars
;

program_declaration: T_PROGRAM T_ID {cline=lineno;}
;

function:
|function_declaration variable_declaration {checkline();} stmts T_RETURN
arithmetic_expression T_SEMI T_ENDFUNCTION
;

function_declaration: T_FUNCTION {checkline();} T_ID T_LPAREN variables T_RPAREN
{cline=lineno;}
;

variables: variable
| variable T_COMMA variables
|
;

variable: type_specifier T_ID array_declaration {putvar($2);}
;

type_specifier: T_INT
| T_CHAR
;

array_declaration:
| T_LBRACK T_NUMBER T_RBRACK
;

variable_declaration: T_VARS {checkline();} variables T_SEMI
|
;

stmts: stmt stmts
| stmt
;

stmt: assignment
| while_stmt
| for_stmt
| if_stmt
| switch_stmt
| print_stmt
| T_BREAK
;

assignment: T_ID T_ASSIGN arithmetic_expression T_SEMI {checkvar($1);}
| T_ID T_ASSIGN T_CHARACT T_SEMI {checkvar($1);}
;

Συντακτικός Αναλυτής - Bison

arithmetic_expression: value
| value arithmetic_operator arithmetic_expression
| T_LPAREN arithmetic_expression T_RPAREN
| T_ADDOP arithmetic_expression
;

value: T_NUMBER
| T_ID array_declaration {checkvar($1);}
;

arithmetic_operator: T_ADDOP
| T_MULOP
| T_RELOP
| T_EQOP
;

while_stmt: T_WHILE T_LPAREN logic_expression T_RPAREN stmts T_ENDWHILE;

logic_expression: arithmetic_expression
| logic_expression T_LOGOP logic_expression
| value T_LPAREN logic_expression T_RPAREN
;

for_stmt: T_FOR T_COUNTER T_COLON T_ASSIGN T_NUMBER T_TO T_NUMBER T_STEP T_NUMBER stmts T_ENDFOR;

if_stmt: T_IF T_LPAREN logic_expression T_RPAREN T_THEN stmts elseif_stmt else_stmt T_ENDIF;

elseif_stmt: T_ELSEIF T_LPAREN logic_expression T_RPAREN stmts elseif_stmt
|
;

else_stmt: T_ELSE stmts else_stm
|
;

switch_stmt: T_SWITCH T_LPAREN logic_expression T_RPAREN T_CASE T_LPAREN logic_expression T_RPAREN T_COLON stmts switch_tail switch_default T_ENDSWITCH;

switch_tail: T_CASE T_LPAREN logic_expression T_RPAREN T_COLON stmts switch_tail
|
;

switch_default: T_DEFAULT T_COLON stmts
|
;

print_stmt: T_PRINT T_LPAREN T_STRING print_tail T_RPAREN T_SEMI;

print_tail: T_LBRACK T_COMMA args T_RBRACK
|
;

args: arg
| arg T_COMMA args
;

arg: T_ID {checkvar($1);}
;

main: T_STARTMAIN {checkline();} variable_declaration stmts T_ENDMAIN;

Συντακτικός Αναλυτής - Bison

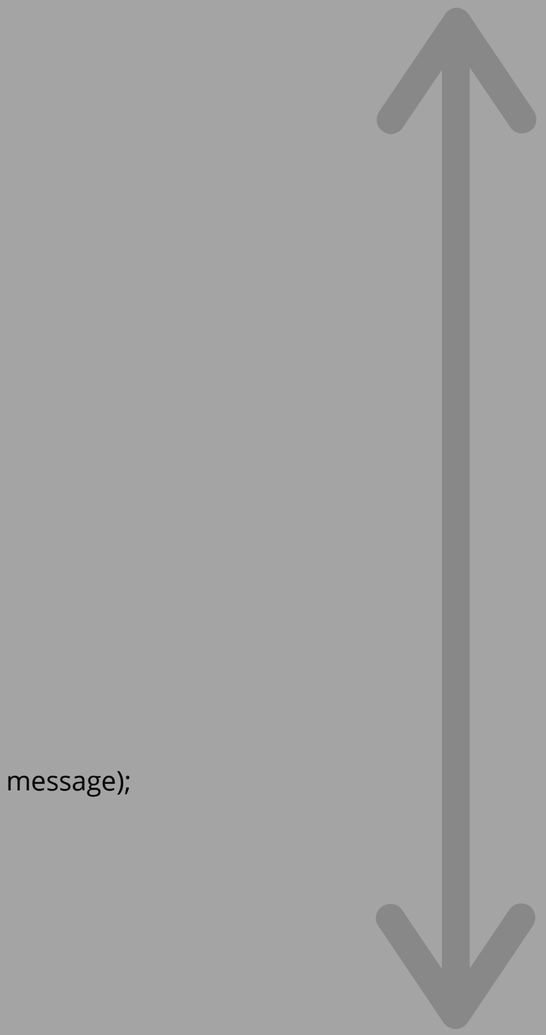
```
%%

int main(int argc, char *argv[])
{
    if(argc > 1){
        yyin = fopen(argv[1], "r");
        if (yyin == NULL){
            perror ("[ERROR] could not open file");
            return EXIT_FAILURE;
        }
    }

    yyparse();
    fclose(yyin);

    return 0;
}

void yyerror(const char *message)
{
    printf("\nError at line %d caused by %s: %s\n", lineno, yytext, message);
}
```



Symbol_table.h αρχείο

```
struct variable
{
  char *name;
  struct variable *next;
};

typedef struct variable variable;
variable *var_table = (variable *)0;

variable *insert ();
variable *search ();

variable *insert ( char *var_name )
{
  variable *ptr;
  ptr = (variable *) malloc (sizeof(variable));
  ptr->name = (char *) malloc (strlen(var_name)+1);
  strcpy (ptr->name,var_name);
  ptr->next = (struct variable *)var_table;
  var_table = ptr;
  return ptr;
}

variable *search ( char *var_name )
{
  variable *ptr;
  for (ptr = var_table; ptr != (variable *)0; ptr = (variable *)ptr->next)
    if (strcmp (ptr->name,var_name) == 0)
      return ptr;
  return 0;
}
```

Συντακτικός Αναλυτής - Bison

Παρατηρήσεις:

- Ο Bison είναι γεννήτορας συντακτικών αναλυτών για γραμματικές χωρίς συμφραζόμενα τύπου LR(1), συνεπώς μπορεί να έχουν συμβεί κάποιες μικρές αλλαγές στην γραμματική που αναπτύχθηκε παραπάνω, προκειμένου να αποφευχθούν shift/reduce ή reduce/reduce conflicts.
- Χρησιμοποιείται η ιδιότητα error-verbose, ώστε να εμφανίζονται αναλυτικά μηνύματα σε περίπτωση συντακτικού λάθους.
- Στα σημεία όπου έπρεπε να ελέγχουμε, αν έχει υπάρξει υποχρεωτική αλλαγή γραμμής (σύμφωνα με την ζητούμενη γραμματική), δυσκολευτήκαμε να βρούμε έναν αποτελεσματικό τρόπο υλοποίησης, αφού το να προσθέσουμε το token '\n' στο αρχείο flex, δημιουργεί μεγάλο πρόβλημα στην υλοποίηση της γραμματικής. Εν τέλει, ορίσαμε μια καινούργια μεταβλητή cline, την οποία εξισώνουμε με την lineno (cline=lineno) κάθε φορά, πριν απ' το token που πρέπει να υπάρχει υποχρεωτική αλλαγή γραμμής. Μετά απ' το token αυτό, ελέγχουμε, αν οι μεταβλητές είναι ίσες (void checkline()). Αν δεν είναι, σημαίνει ότι συνέβη αλλαγή γραμμής, διαφορετικά εμφανίζεται κατάλληλο μήνυμα και σταματά η εκτέλεση του προγράμματος.
- Όσον αφορά στον έλεγχο της δήλωσης μεταβλητών (3ο ερώτημα της εργασίας), δημιουργήσαμε έναν hash table, στον οποίο εισάγουμε κάθε φορά, όταν γίνεται δήλωση μεταβλητής στο πρόγραμμα, το όνομα αυτής (void putvar(char *var_name)). Στη συνέχεια, κάνουμε έλεγχο σε κάθε αριθμητική ή λογική έκφραση που υπάρχει στο πρόγραμμα, αν οι μεταβλητές που χρησιμοποιούνται, υπάρχουν στον πίνακα (άρα και έχουν δηλωθεί). Αν δεν υπάρχουν (void checkvar(char *var_name)), τότε εμφανίζεται κατάλληλο μήνυμα και σταματά η εκτέλεση του προγράμματος. Το αρχείο που υλοποιεί τον hash table έχει συμπεριληφθεί στον bison μέσω της εντολής include και είναι το symbol_table.h.
- Χρησιμοποιουμε και τη %union, για να "ορίσουμε" ουσιαστικά τον τύπο κάποιων απ' των token (int και string), ώστε να μπορούν να γίνονται πράξεις με τη λειτουργία $ που υποστηρίζει ο bison αλλά και να αποθηκεύονται τα token στην μνήμη, σύμφωνα με τον τύπο δεδομένων τους.

Παραδείγματα

(μέσω screenshots)

```
PROGRAM project
STRUCT var45
VARS
INT mar1;
CHAR martios[5];
ENDSTRUCT

FUNCTION foo(INT var1)
IF (var1<-5) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1 = var1 - 1;
ENDFOR
ELSE
var1 = 1;
ENDIF
RETURN 0;
END_FUNCTION

STARTMAIN

VARS CHAR var2;
var2 = 5;

PRINT("marios einai" [,var1, var2

FOR counter:=1 TO 5 STEP 1
var1 = var1 - 1;
ENDFOR

ENDMAIN
```

```
DELL VOSTRO V131@DESKTOP-1D79NEP ~
$ ./a.exe test
PROGRAM project
STRUCT var45
VARS
INT mar1;
CHAR martios[5];
ENDSTRUCT

FUNCTION foo(INT var1)
IF (var1<-5) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1 = var1 - 1;
ENDFOR
ELSE
var1 = 1;
ENDIF
RETURN 0;
END_FUNCTION

STARTMAIN

VARS CHAR var2;
var2 = 5;

PRINT("marios einai" [,var1, var2]);

FOR counter:=1 TO 5 STEP 1
var1 = var1 - 1;
ENDFOR

ENDMAIN
```

Στο screenshot αριστερά παρατηρούμε το αρχείο εισόδου (-> test) και δεξιά, αφού εκτελέσουμε τις κατάλληλες εντολές, επιστρέφεται το αποτέλεσμα, το οποίο στην συγκεκριμένη περίπτωση είναι το πρόγραμμα, πράγμα που σημαίνει ότι δεν υπάρχει κάποιο λάθος στην σύνταξη του.

Χρησιμοποιήσαμε τις παρακάτω εντολές:
flex compiler.l
bison -v -d bison.y
gcc bison.tab.c lex.yy.c -lm
./a.exe test

Παραδείγματα

(μέσω screenshots)

```
PROGRAM project

STARTMAIN

VARS INT var2;

IF (var1<-5 AND var2==2) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR
ELSE
var1= 1;
ENDIF

ENDMAIN
```

```
$ ./a.exe test
PROGRAM project

STARTMAIN

VARS INT var2;

IF (var1<       var1 is an undeclared identifier.
```

Παρατηρούμε ότι στο πρόγραμμα χρησιμοποιείται η μεταβλητή var1, η οποία δεν έχει δηλωθεί προηγουμένως. Για το λόγο αυτό, μόλις την "συναντήσει", εμφανίζει κατάλληλο μήνυμα και τερματίζει την εκτέλεση του προγράμματος.

Η εργασία υλοποιήθηκε μέσω
Cygwin, με τη χρήση του Visual Studio Code.

Παραδείγματα

(μέσω screenshots)

```
PROGRAM project STARTMAIN

VARS INT var2, INT var1;

IF (var1<-5 AND var2==2) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR
ELSE
var1= 1;
ENDIF

ENDMAIN
```

```
DELL VOSTRO V131@DESKTOP-1D79NEP ~
$ ./a.exe test
PROGRAM project STARTMAIN
Error at line 1 caused by STARTMAIN: new line is required

DELL VOSTRO V131@DESKTOP-1D79NEP ~
$
```

Σύμφωνα με την γραμματική μετά από το ID του προγράμματος, θα πρέπει να υπάρχει υποχρεωτική αλλαγή γραμμής. Παρατηρούμε λοιπόν πως αυτό δεν συμβαίνει, συνεπώς κατά την εκτέλεσή του προγράμματος, εμφανίζεται κατάλληλο μήνυμα και διακόπτεται η εκτέλεση του (αφού θεωρείται συντακτικό λάθος).

Παραδείγματα

(μέσω screenshots)

```
PROGRAM project

 STARTMAIN

VARS INT var2, INT var1; % auto einai ena sxolio

IF (var1<-5 AND var2==2) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR
ELSE
var1= 1;
ENDIF

ENDMAIN
```

```
PROGRAM project

STARTMAIN

VARS INT var2, INT var1; /* auto einai ena
                            multiline comment */

IF (var1<-5 AND var2==2) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR
ELSE
var1= 1;
ENDIF

ENDMAIN
```

Δεξιά, παρατηρούμε ότι εξαιτίας λανθασμένης σύνταξης του σχολίου πολλαπλών γραμμών, κατά τη διάρκεια της εκτέλεσης, εμφανίζεται σφάλμα και διακόπτεται η εκτέλεση του προγράμματος.

```
PROGRAM project

STARTMAIN

VARS INT var2, INT var1;

IF (var1<-5 AND var2==2) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR
ELSE
var1= 1;
ENDIF

ENDMAIN
```

```
PROGRAM project

STARTMAIN

VARS INT var2, INT var1; /* auto einai ena
                            multiline comment

IF (var1<-5 AND var2==2) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR
ELSE
var1= 1;
ENDIF

ENDMAIN
```

Αριστερά, παρατηρούμε την επιτυχή δημιουργία σχολίων μίας αλλά και πολλαπλών γραμμών.

```
DELL VOSTRO V131@DESKTOP-1D79NEP ~
$ ./a.exe test
PROGRAM project

STARTMAIN

VARS INT var2, INT var1;
Error at line 18 caused by : syntax error, unexpected T_EOF

DELL VOSTRO V131@DESKTOP-1D79NEP ~
$
```

Παραδείγματα

(μέσω screenshots)

```
st
  PROGRAM project
  TYPEDEF STRUCT var45
  VARS
  INT mar1;
  CHAR martios[5];
  var45 ENDSTRUCT

  FUNCTION example(INT var1)
  IF (var1<-5) THEN
  var1=0;
  FOR counter:=1 TO 5 STEP 1
  var1= var1- 1;
  ENDFOR
  ELSE
  var1= 1;
  ENDIF
  RETURN 0;
  END_FUNCTION

  STARTMAIN

  VARS CHAR var2;

  PRINT("marios einai" [,var1, var2]);

  FOR counter:=1 TO 5 STEP 1
  var1= var1- 1;
  ENDFOR

  ENDMAIN
```

Αριστερά, παρατηρούμε μια επιτυχημένη δήλωση struct, αφού ακολουθεί τους κανόνες της γραμματικής

```
$ ./a.exe test
PROGRAM project
TYPEDEF STRUCT var45
VARS
INT mar1;
CHAR martios[5];
var45 ENDSTRUCT

FUNCTION example(INT var1)
IF (var1<-5) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR
ELSE
var1= 1;
ENDIF
RETURN 0;
END_FUNCTION

STARTMAIN

VARS CHAR var2;

PRINT("marios einai" [,var1, var2]);

FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR


ENDMAIN


DELL VOSTRO V131@DESKTOP-1D79NEP ~
$
```

Δεξιά, παρατηρούμε μια αποτυχημένη δήλωση struct, με αποτέλεσμα να εμφανίζεται κατάλληλο μήνυμα και να διακόπτεται η εκτέλεση του προγράμματος

```
PROGRAM project
TYPEDEF STRUCT var45
VARS
INT mar1;
CHAR martios[5];
var45

FUNCTION example(INT var1)
IF (var1<-5) THEN
var1=0;
FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR
ELSE
var1= 1;
ENDIF
RETURN 0;
END_FUNCTION

STARTMAIN

VARS CHAR var2;

PRINT("marios einai" [,var1, var2]);

FOR counter:=1 TO 5 STEP 1
var1= var1- 1;
ENDFOR

ENDMAIN
```

```
DELL VOSTRO V131@DESKTOP-1D79NEP ~
$ ./a.exe test
PROGRAM project
TYPEDEF STRUCT var45
VARS
INT mar1;
CHAR martios[5];
var45

FUNCTION
Error at line 8 caused by FUNCTION: syntax error, unexpected FUNCTION, expecting ENDSTURCT

DELL VOSTRO V131@DESKTOP-1D79NEP ~
$
```


ΤΕΛΟΣ ΕΡΓΑΣΙΑΣ