ΑΝΑΦΟΡΑ ΕΞΑΜΗΝΙΑΙΑΣ ΕΡΓΑΣΙΑΣ ΣΤΟ ΜΑΘΗΜΑ
“ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΓΛΩΤΙΣΤΩΝ”

Περιεχόμενα

Ερώτημα 1ο

BNF

<doc> ::= PROG <pname><wrapper><struct_wrapper><func><main> | <wrapper><doc>
<wrapper> ::= <comment> | <EOL>
<end_wrapper> ::= є | <wrapper>
<pname> ::= <letter> | <digit> | ”_” | <letter><pname> | <digit><pname> | ”_”<pname>
<fname> ::= <letter> | ”_” | <letter><pname> | ”_”<pname>
<vname> ::= <letter> | ”_” | <letter><pname> | ”_”<pname>
<vartype> ::= <character> | <integer> | <fname>
<character> ::= <letter> | <digit> | <symbol>
<array> ::= є | “ [“ <digit> “] “ <array>
<number> ::= <digit> | “-”<digit> | <digit><number> | “-”<digit><number>
<num_oper> ::= “*” | “/” | “+” | “-” | “^”

<struct_wrapper> ::= є | STRUCT <struct> ESTRUCT <wrapper><struct_wrapper> | TYPEDEF STRUCT <struct> ESTRUCT <vname>
<struct> ::= <vname><wrapper>VARS<struct_vars>
<struct_vars> ::= <vartype> <vname> <arr> <append_struct_vars> | <wrapper><vartype> <vname> <arr> <append_struct_vars>
<append_struct_vars> ::= “;”<wrapper> | “;”<struct_vars> | “,”<vname><arr><append_struct_vars>
<func> ::= є | <function> FUNC <fname> ”(“ <func_param><wrapper><func_body> RET <return_type><wrapper> ENDFUNC <wrapper>
<main> ::= SMAIN <wrapper><func_body> EMAIN <end_wrapper>
<func_param> ::= є | <vartype><vname><arr><append_func_param>
<append_func_param> ::= є | “,” <func_param>
<func_arg> ::= є | <letter><arr><append_func_arg> | <number><append_func_param> | “_” <arr><append_func_arg> | <fname><arr><append_func_arg>
<append_func_arg> ::= є | “,” <func_arg>

<func_body> ::= VARS<wrapper><func_var> | <commands>
<func_var> ::= <vartype><vname><array><append_func_vars>
<append_func_vars> ::= “;”<func_var> | “;”<wrapper><func_var> |
“;”<commands> | “,”<vname><array><append_func_vars>
<return_type> ::= <vname> | <number> | <bool>

<commands> ::= ϵ | <wrapper><commands> | <assign><commands> |
<loop><commands> | <flow_control><commands> | <print><commands>
<extend_commands> ::= ϵ | <wrapper><extend_commands> |
<assign><extend_commands> |
<loop><extend_commands> | <flow_control><extend_commands> |
<print><extend_commands> | BREAK “;” <extend_commands>

<assign> ::= <vname> | “=” <rvalue> “;”
<rvalue> ::= <vname><append_rvalue> | <number><append_rvalue> |
“(” <rvalue> “)” <append_rvalue> | <fname> “(” <func_arg> “)” <append_rvalue>
<append_rvalue> ::= ϵ | <num_oper><rvalue>

<loop> ::= <while> | <for>
<while> ::= WHILE “(“ <cond> “)” <wrapper><extend_commands> EWHILE
<cond> ::= <rvalue> | <rvalue><comp_oper><cond> |
<rvalue><log_oper><cond>
<for> ::= FOR <vname> “:=” <number> TO <number> STEP
<number><wrapper><extend_commands> EFOR

<flow_control> ::= <if> | <switch>
<if> ::= IF <cond> THEN <wrapper><extend_commands> ENDIF
<else> ::= ϵ | ELSEIF <wrapper><extend_commands><else> |
ELSE <wrapper><extend_commands>
<switch> ::= SWITCH “(” <rvalue> “)” “:” <wrapper><commands><case>
<case> ::= CASE “(” <rvalue> “)” “:” <wrapper><extend_commands><case> |
DEFAULT “:” <extend_commands> ESWITCH | ESWITCH

<print> ::= PRINT “(” <fname><print_var> “)” “;”
<print_var> ::= ϵ | “,”<vname><print_var>

<letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<symbol> ::= "|" | " " | "!" | "#" | "$" | "%" | "&" | "(" | ")" | "*" | "+" | "," | "-" | "." | "/" | ":" | ";" | ">" | "=" | "<" | "?" | "@" | "[" | "\" | "]" | "^" | "_" | "`" | "{" | "}" | "~"
<bool> ::= “0” | “1”
<comp_oper> ::= “<” | “>” | “!=” | “==”
<log_oper> ::= “&”“&” | “|”“|”

Screenshot ορθής και λανθασμένης λειτουργίας

Screenshot για την λειτουργία των εντολών ανάθεσης.

α. Ορθή λειτουργία

```
1   PROGRAM test1
2
3   FUNCTION test()
4
5   VARS
6       CHAR var1;
7       var1 = 1;
8
9   RETURN var1
10  END_FUNCTION
11
12  STARTMAIN
13
14  ENDMAIN
15
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample1.txt
Scope: main
arr_of_scopes[0] = test
         var1,
         0 arguments with types:
arr_of_scopes[1] = main

         0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$
```

β. Λανθασμένη λειτουργία

```
1   PROGRAM test1
2
3   FUNCTION test()
4
5   VARS
6       CHAR var1;
7       var1 == 1;
8
9   RETURN var1
10  END_FUNCTION
11
12  STARTMAIN
13
14  ENDMAIN
15
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample1.txt
error in line 7: syntax error, unexpected COMP_OPER, expecting ALPHA or UNDER or
 SAU
manolis@manoslaptop:~/Desktop/arxes$
```

Screenshot για την λειτουργία των εντολών βρόγχου.

WHILE – ENDWHILE

α. Ορθή λειτουργία

```
1   PROGRAM test1
2
3   STARTMAIN
4   VARS
5       INTEGER var1[5];
6       INTEGER var2;
7       var2=7;
8
9       WHILE(var2<7 OR var2==7)
10      %comment
11
12      ENDWHILE
13
14
15  ENDMAIN
16
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample1.txt
Scope: main
arr_of_scopes[0] = main
        var1,   var2,
        0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$
```

β. Λανθασμένη λειτουργία

```
1   PROGRAM test1
2
3   STARTMAIN
4   VARS
5       INTEGER var1[5];
6       INTEGER var2;
7       var2=8;
8
9   WHILE(var2<7 OR var2=8)
10      %comment
11
12  ENDWHILE
13
14  ENDMAIN
15
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample1.txt
error in line 9: syntax error, unexpected EQS, expecting CP
manolis@manoslaptop:~/Desktop/arxes$
```

FOR-ENDFOR

α. Ορθή λειτουργία

```
1   PROGRAM test1
2
3   STARTMAIN
4   VARS
5       INTEGER var1[5];
6       INTEGER var2;
7       var2=8;
8
9
10  FOR var2 := 1 TO 8 STEP 1
11      var1=var2;
12  ENDFOR
13
14  ENDMAIN
15
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample1.txt
Scope: main
arr_of_scopes[0] = main
        var1,   var2,
        0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$
```

β. Λανθασμένη λειτουργία

```
1   PROGRAM test1
2
3   STARTMAIN
4   VARS
5       INTEGER var1[5];
6       INTEGER var2;
7       var2=8;
8
9
10  FOR var2 = 1 TO 8 STEP 1
11      var1=var2;
12  ENDFOR
13
14  ENDMAIN
15
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample1.txt
error in line 10: syntax error, unexpected EQS, expecting COLON_EQ
manolis@manoslaptop:~/Desktop/arxes$
```

Screenshot για την λειτουργία των εντολών ελέγχου

IF - ELSEIF - ELSE - ENDIF

α. Ορθή λειτουργία

```
PROGRAM test2

STARTMAIN
VARS
    INTEGER var1[5];
    INTEGER var2;
    var2=8;

IF var2==8 THEN
    var1=4;
ELSE
    var1=5;
ENDIF

ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample2.txt
Scope: main
arr_of_scopes[0] = main
        var1,   var2,
        0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$
```

β. Λανθασμένη λειτουργία

```
PROGRAM test2

STARTMAIN
VARS
    INTEGER var1[5];
    INTEGER var2;
    var2=8;

IF var2==7 THEN
    var1=4;
ELSE 2<3
    var1=5;
ENDIF

ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample2.txt
error in line 11: syntax error, unexpected POS_NUM, expecting EOL or COMMENT
manolis@manoslaptop:~/Desktop/arxes$
```

SWITCH - CASE – ENDSWITCH

α. Ορθή λειτουργία

```
1   PROGRAM test3
2
3   STARTMAIN
4   VARS
5       INTEGER var1;
6       INTEGER var2;
7       var1=5;
8       var2=8;
9
10  SWITCH(var2)
11  %comment
12  CASE(var1):
13      var1=var2;
14  CASE(var2):
15      var2=var1;
16  ENDSWITCH
17
18  ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample3.txt
Scope: main
arr_of_scopes[0] = main
        var1,   var2,
        0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$
```

β. Λανθασμένη λειτουργία

```
1   PROGRAM test3
2
3   STARTMAIN
4   VARS
5       INTEGER var1;
6       INTEGER var2;
7       var1=5;
8       var2=8;
9
10  SWITCH(var2)
11  %comment
12  CASE(var1)
13      var1=var2;
14  CASE(var2):
15      var2=var1;
16  ENDSWITCH
17
18  ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample3.txt
error in line 13: syntax error, unexpected EOL, expecting COLON
manolis@manoslaptop:~/Desktop/arxes$
```

Screenshot για την λειτουργία της εντολής εκτύπωσης στην οθόνη.

α. Ορθή λειτουργία

```
1   PROGRAM test2
2
3   STARTMAIN
4   VARS
5       INTEGER var1;
6       INTEGER var2;
7       var1=5;
8       var2=8;
9
10  IF var2==8 THEN
11      PRINT("The value of variable 1 is:",var1);
12  ELSE
13      %comment
14  ENDIF
15
16  ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample2.txt
Scope: main
arr_of_scopes[0] = main
        var1,   var2,
        0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$
```

β.Λανθασμένη λειτουργία

```
1   PROGRAM test2
2
3   STARTMAIN
4   VARS
5       INTEGER var1;
6       INTEGER var2;
7       var1=5;
8       var2=8;
9
10  IF var2==8 THEN
11      PRINT("The value of variable 1 is:,var1);
12  ELSE
13      %comment
14  ENDIF
15
16  ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
        var1,   var2,
        0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample2.txt
Unused character: "
error in line 11: syntax error, unexpected ALPHA, expecting STR
manolis@manoslaptop:~/Desktop/arxes$
```

Screenshot για την λειτουργία της εντολής τερματισμού βρόγχου.

α. Ορθή λειτουργία

```
PROGRAM test3

STARTMAIN
VARS
    INTEGER var1;
    INTEGER var2;
    var1=5;
    var2=8;

SWITCH(var2)
%comment
CASE(var1):
    var1=var2;
CASE(var2):
    var2=var1;
    BREAK;
ENDSWITCH

ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample3.txt
Scope: main
arr_of_scopes[0] = main
        var1,   var2,
        0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$
```

β. Λανθασμένη λειτουργία

```
PROGRAM test3

STARTMAIN
VARS
    INTEGER var1;
    INTEGER var2;
    var1=5;
    var2=8;

SWITCH(var2)
%comment
CASE(var1):
    var1=var2;
CASE(var2):
    var2=var1;
    BREAK;
ENDSWITCH

ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample3.txt
Scope: main
arr_of_scopes[0] = main
        var1,   var2,
        0 arguments with types:
manolis@manoslaptop:~/Desktop/arxes$
```

Ερώτημα 2ο

Screenshot για την λειτουργία struct.

α. Ορθή λειτουργία

```
PROGRAM test

%This is my first struct
STRUCT s1
VARS
    CHAR str[20], c;
    INTEGER i; INTEGER j;
ENDSTRUCT %the end of my first struct

%My struct may start with an underscore
%and may also have two names!
TYPEDEF STRUCT _super_complicated
VARS
    _super_complicated part1[2];
    s1 part2;
ENDSTRUCT s2

%TODO: do not allow a variable name to be used twice
FUNCTION func(INTEGER arg1, s2 arg2)
VARS
    INTEGER i;
    s2 struct_f;

%recursive func call
i = func(i, struct_f);

%if return a variable must be declared
RETURN i
END_FUNCTION

STARTMAIN
VARS
    INTEGER maini; CHAR c;

maini = -20;
c = func(maini, c);
ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample.txt
Scope: main
arr_of_scopes[0] = func
        arg1,   arg2,   i,      struct_f,
        2 arguments with types:
        INTEGER
        s2
arr_of_scopes[1] = main
        maini,  c,
        0 arguments with types:
struct_names[0] = s1
struct_names[1] = _super_complicated
struct_names[2] = s2
manolis@manoslaptop:~/Desktop/arxes$
```

β.Λανθασμένη λειτουργία

```
PROGRAM test

%This is my first struct
STRUCT s1
VARS
    CHAR str[20], c;
    INTEGER i; INTEGER j;
ENDSTRUCT %the end of my first struct

%My struct may start with an underscore
%and may also have two names!
STRUCT _super_complicated
VARS
    _super_complicated part1[2];
    s1 part2;
ENDSTRUCT s2

%TODO: do not allow a variable name to be used twice
FUNCTION func(INTEGER arg1, s2 arg2)
VARS
    INTEGER i;
    s2 struct_f;

%recursive func call
i = func(i, struct_f);

%if return a variable must be declared
RETURN i
END_FUNCTION

STARTMAIN
VARS
    INTEGER maini; CHAR c;

maini = -20;
c = func(maini, c);
ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample.txt
error in line 16: syntax error, unexpected SAU, expecting EOL or COMMENT
manolis@manoslaptop:~/Desktop/arxes$
```

<u>Σύμβαση</u>: αντί για σύνταξη (<όνομα τύπου> ENDSTRUCT) υλοποιήσαμε την (ENDSTRUCT <όνομα τύπου>).

Ερώτημα 3ο

Screenshot για τον έλεγχο σωστής δήλωσης μεταβλητών που χρησιμοποιούνται οπουδήποτε στο πρόγραμμα.

α.Ορθή λειτουργία.

Φαίνεται από το ερώτημα 2.

β. Λανθασμένη λειτουργία

```
30
31  STARTMAIN
32  VARS
33      INTEGER maini; CHAR c;
34
35  main = -20;
36  c = func(maini, c);
37  ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample.txt
error in line 35: Variable is not declared in this scope!
manolis@manoslaptop:~/Desktop/arxes$
```

Screenshot για τον έλεγχο ορθής κλήσης των συναρτήσεων.

α. Ορθή λειτουργία.

Φαίνεται από το ερώτημα 2.

β. Λανθασμένη λειτουργία

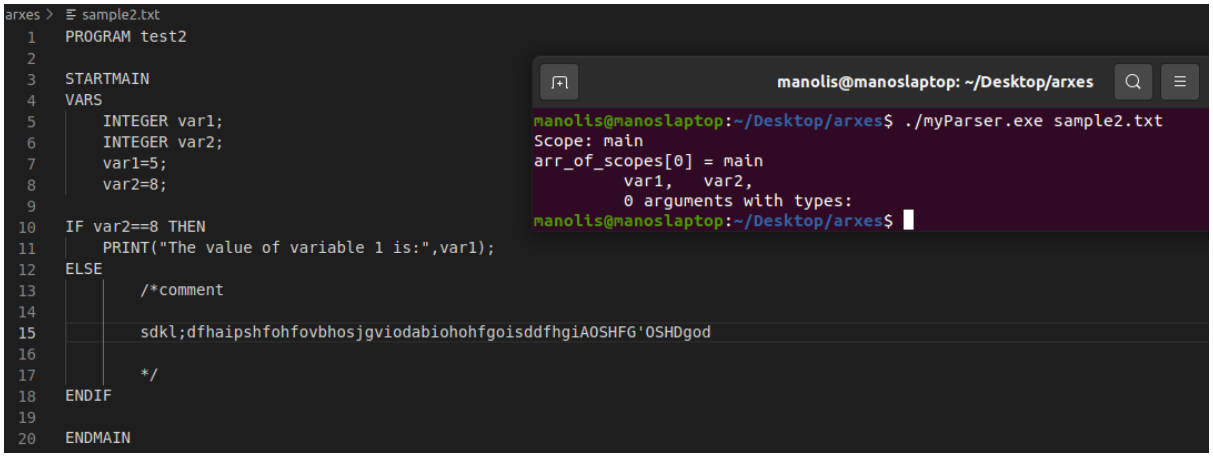
```
19  FUNCTION func(INTEGER arg1, s2 arg2)
20  VARS
21      INTEGER i;
22      s2 struct_f;
23
24  %recursive func call
25  i = func(i, struct_f);
26
27  %if return a variable must be declared
28  RETURN i
29  END_FUNCTION
30
31  STARTMAIN
32  VARS
33      INTEGER maini; CHAR c;
34
35  maini = -20;
36  c = funcc(maini, c);
37  ENDMAIN
```

```
manolis@manoslaptop: ~/Desktop/arxes
manolis@manoslaptop:~/Desktop/arxes$ ./myParser.exe sample.txt
error in line 36: Function is not declared!
manolis@manoslaptop:~/Desktop/arxes$
```

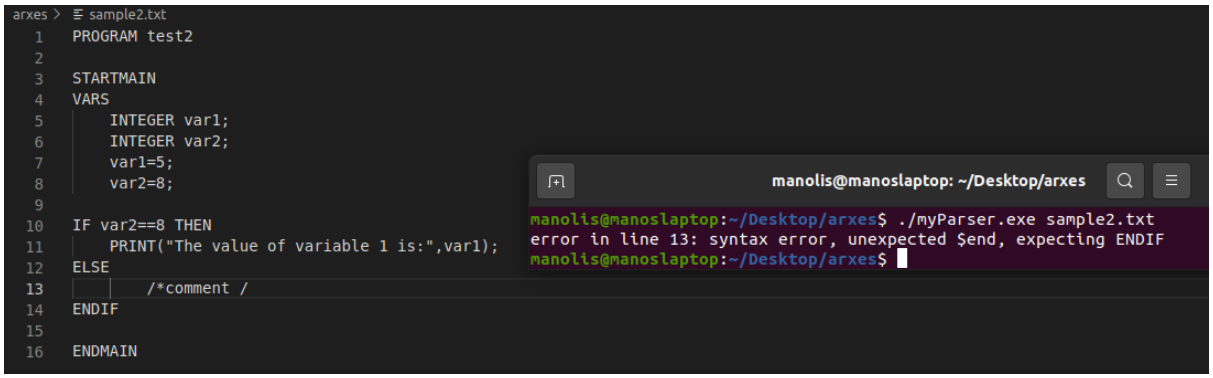
Ερώτημα 4ο

Screenshot για την λειτουργία των σχολίων πολλαπλών γραμμών.

α. Ορθή λειτουργία



β. Λανθασμένη λειτουργία




```
"/*"    {BEGIN(c_comment);}
<c_comment>"*/"          {BEGIN(0);}
<c_comment>(\n)+|.       { }
```

Ο τρόπος με τον οποίο επιλέξαμε να διαχειριστούμε τα σχόλια πολλαπλών γραμμών, χωρίς τη χρήση Regular Expression ήταν μέσω της δυνατότητας του flex να αλλάξει κατάσταση. Η ιδέα είναι ότι από τη στιγμή που θα εντοπιστεί σε οποιοδήποτε σημείο του κώδικα το “/*” μεταβαίνουμε στην κατάσταση c_comment η οποία στην πραγματικότητα αγνοεί όλα τα σύμβολα, μέχρι να εντοπίσει το σύμβολο τερματισμού των multiline comment (“*/”).

Είσοδος στο flex - αρχείο scanner.l

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

   int line = 1;
   int is_tab(char c[4]);
   int count_newlines(char* token);

%}
%x c_comment
eol [\n]+
whitespace [ \t\n]*

alpha [a-zA-Z]
digit [0-9]
under [_]

start_au ({alpha}|{under})+({alpha}|{under}|{digit})*
anu {alpha}|{digit}|{under}

add "+"
sub "-"
mul "*"
div "/"
pow "^"

gt ">"
lt "<"
neq "!="
is_eq "=="
comp_operator {gt}|{lt}|{neq}|{is_eq}

and "AND"
or "OR"
log_operator {and}|{or}

str \".*\"
true "TRUE"
false "FALSE"
bool {true}|{false}

comment {whitespace}?"%".*{eol}?

%%
"PROGRAM" {return PROG;}
"RETURN" {return RET;}
"FUNCTION" {return FUNC;}
"END_FUNCTION" {return ENDFUNC;}
```

```
"STARTMAIN" {return SMAIN;}
"ENDMAIN" {return EMAIN;}
"CHAR" {strcpy(yylval.str, yytext); return CHAR;}
"INTEGER" {strcpy(yylval.str, yytext); return INT;}
"VARS" {return VARS;}
"WHILE" {return WHILE;}
"ENDWHILE" {return EWHILE;}
"FOR" {return FOR;}
"TO" {return TO;}
"STEP" {return STEP;}
"ENDFOR" {return EFOR;}
"IF" {return IF;}
"THEN" {return THEN;}
"ELSEIF" {return ELSEIF;}
"ELSE" {return ELSE;}
"ENDIF" {return ENDIF;}
"SWITCH" {return SWITCH;}
"CASE" {return CASE;}
"DEFAULT" {return DEFAULT;}
"ENDSWITCH" {return ESWITCH;}
"PRINT" {return PRINT;}
"BREAK" {return BREAK;}
"STRUCT" {return STRUCT;}
"ENDSTRUCT" {return ESTRUCT;}
"TYPEDEF" {return TYPEDEF;}

"/*"   {BEGIN(c_comment);}
<c_comment>"*/"        {BEGIN(0);}
<c_comment>(\n)+|.      { }

{add} {return ADD;}
{sub} {return SUB;}
{mul} {return MUL;}
{div} {return DIV;}
{pow} {return POW;}

{comp_operator} {return COMP_OPER;}
{log_operator} {return LOG_OPER;}

":" {return COLON;}
":=" {return COLON_EQ;}
"," {return COMMA;}
";" {return SEMIC;}
"=" {return EQS;}
{comment}+ {line += count_newlines(yytext); return COMMENT;} /*the ? at the end of the regex is
there in order to
                    allow flex scann EOF even though there is a comment on the last line.
                    (another solution would be for each code that contains a comment at the end of
the line,
                    have an empty line after the comment)
*/
```

```
{eol} {line += strlen(yytext); return EOL;} //If eof is expected, you are allowed to \n as many times as
you want
"(" {return OP;}
")" {return CP;}
"[" {return OB;}
"]" {return CB;}
[ ] /*Ignore Whitespace*/

{bool} {return BOOL;}
{alpha}+ {strcpy(yylval.str, yytext); return ALPHA;}
{digit}+ {yylval.ival = atoi(yytext); return POS_NUM;}
{under}+ {strcpy(yylval.str, yytext); return UNDER;}
{start_au}+ {strcpy(yylval.str, yytext); return SAU;} //Starting with an Alpha or Under
{anu}+ {return ANU;} //Alpha Number Under
{str} {return STR;}
. { printf("Unused character: %c\n", *yytext); }


%%

int count_newlines(char* token){
    int cnt = 0, i;
    for(i=0;i<strlen(token);i++){
        if(token[i] == '\n'){
            cnt++;
        }
    }
    //printf("Counted %d after comment newlines\n",cnt);
    //printf("i has a value of: %d\n",i);
    return cnt;
}

/*//IF YOU NEED TO CHANGE THE BEHAVIOUR OF HOW A WHITESPACE IS TREATED THIS
FUNCTION MUST ALSO CHANGE (and the flex rule)!!!
//tab will match any space that is at the beggining of a line
//if there are 4 spaces in a row (that is considered to be a tab)
//return 1 (so flex returns the TAB token)
//if there are not enough spaces, then return 0, which will lead flex ignoring them
//TODO: There should be a better way of doing this! FIX IT
int is_tab(char c[4]){
    int i, counter = 0;
    for (i = 0 ; i < 4; i++){
        if(c[i] == ' '){
            counter++;
        }
    }
    if(counter == 4){
        return 1;
    }
    return 0;
}
```

if you uncomment this, make sure you add the line below, before the whitespace ignore line
^[\t]+ { if(is_tab(yytext)){ return TAB; } }*/

Είσοδος στο Bison -αρχείο parser.y

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

//set to 1 if in debug mode
         #define DEBUG 1

//maximum amount of allowed structs
         #define MAX_STRUCTS 30

//maximum length of a struct/function name
//if you change this, make sure there is enough space in yylval array of characters
         #define MAX_NAME 50

//maximum number of variables defined in a function/struct/main (including arguments)
         #define MAX_NUM_of_VARS 50

//maximum number of arguments in a function
         #define MAX_ARGS 10

//maximum number of allowed scopes (function, main)
         #define MAX_SCOPE 20
   int yylex();
   void yyerror(const char *s);

//my functions

         //checks if i is grater that zero
         void isGTZ(int i);

         //checks if str has been defined as struct
         int isStruct(char str[MAX_NAME]);

         //check if str is a valid type for a variable
         int isVartype(char str[MAX_NAME]);

         //adds the name (str) of a struct to the global array (struct_names)
         void addStruct_name(char* str);

         //used for checking if a var is declaired (if it is return 1 else return 0)
         //if error_msg not used, pass ""
         int is_declaired(char* var, char* error_msg);

         //used for checking if a func is declaired (if it is return 1 else return 0)
         //if error_msg not used, pass ""
         int is_function(char* func, char* error_msg);
```

```
	void add_in_cur_scope(char* var);

	//called on function definition. It will save the type (ex. INT) of the parameters.
	//TODO: use this to check if arguments are of correct type
	void add_type_of_arg(char* type);

	//Checks if the call of a function has the right amount of arguments (using arg_cnter)
	//if not, returns an error
	int is_argument_count_correct(char* func_name);

	//GLobal array to save the struct names in
	char struct_names[MAX_STRUCTS][MAX_NAME];

	//No idea, may be redundant
	char strct[MAX_NAME];

	//Used for finding the correct position to add a new struct name in struct_names array
	//Maybe it is a good idea to update some functions not to use this!
	int pos = 0;

	//may be redundant
	char cur_scope[MAX_NAME];

	//Organize each scope in a struct
	struct scope{
		//name of function not more than 50 char
		char scope[MAX_NAME];
		//max 50 variables (including arguments) allowed in a scope
		char vars[MAX_NUM_of_VARS][MAX_NAME];
		//counter of arguments
		int argc;
		//no more than 10 arguments allowed (each arg type not more than 50 char)
		char argt[MAX_ARGS][MAX_NAME];
	};

	//An array of structs. Maybe have the size be a define
	struct scope arr_of_scopes[MAX_SCOPE];

	//maybe update functions to find the cur_scope and delete this
	int scope_cnt = 0;

	//argument counter to check if function called with the right amount of args
	int arg_cnter = 0;
%}

%define parse.error detailed //DO I NEED TO HAVE CUSTOM MESSAGES????


//yylval union
%union
{
	char str[50];
```

```
        int ival;
}

//Commited KeyWORDS
%token PROG //PROGRAM
%token STRUCT ESTRUCT TYPEDEF //STRUCTS
%token FUNC ENDFUNC //FUNCTIONS
%token SMAIN EMAIN //MAIN
%token <str> VARS CHAR INT //VARIABLES (VARS does not really have a type but oh well)
%token WHILE EWHILE //WHILE
%token FOR TO STEP EFOR //FOR
%token IF THEN ELSEIF ELSE ENDIF //IF
%token SWITCH CASE DEFAULT ESWITCH //SWITCH
%token PRINT BREAK //PRINT and BREAK (break only allowed inside a loop or a flow control
statement)

//Symbols
%token EOL RET COMMENT //End Of Line RETurn
%token OP CP OB CB //Open Parenthnsis Close Parentensis Open Brackets Close Brackets
%token ADD SUB MUL DIV POW //(+) (-) (/) (*) (^)
%token COMMA SEMIC EQS COMP_OPER LOG_OPER COLON_EQ //(,) (;) (=) (< > != ==) (AND
OR) (:=)
%token COLON //(:)

//Tokens that may use the union (yylval)
//<union variable> TOKEN
%token <ival> POS_NUM // assings token NUM to have a type of ival
%token BOOL
%token <str> ALPHA UNDER ANU SAU STR

//%type for assigning a union field to a rule
%type <str> vartype vname fname;
%type <ival> num;
%token FLOAT

//BE EXTREMELY CAREFUL ON HOW ACTIONS ARE EXECUTED!!!!!
//FIRST ARE THE LEAFS AND LAST THE ROOT
//ON YOUR PREVIOUS IMPLEMENTATION YOU HAD:

// doc: PROG pname wrapper structs{ printf("Program set!\n\n"); }
//          | wrapper doc;

// structs: STRUCT sname wrapper ESTRUCT wrapper {addStruct_name($2);}
//          | func_main;

//THIS COULD NEVER WORK SINCE FIRST WILL BE EXCECUTED func_main's ACTIONS
//AND THEN THE GLOBAL ARRAY WOULD BE UPDATED WITH THE STRUCT NAMES!!!

//WARNING!!!
//THE SAME PROBLEM WILL APPLY TO THE FUNCTION DEFINITIONS!!!

%%
```

```
//starting rule
doc: PROG pname
	{
		strcpy(cur_scope, "global");
	}
	wrapper struct_wrapper func main
	| wrapper doc;

//wrapper used when a newline is required
//COMMENT is implemented in a way that it will
//also accept at least one \n
wrapper: COMMENT
	| EOL;

//wrapper used for end of program (after ENDMAIN)
end_wrapper: %empty
	| wrapper;

//program name, may even begin with a number
pname: ALPHA
	| POS_NUM
	| UNDER
	| SAU
	| ANU;

//function name, must not start with a number
fname: ALPHA
	| UNDER
	| SAU;

//variable name, could be merged with fname (vname is also used as a struct name)
vname: ALPHA
	| UNDER
	| SAU;

//all the valid variable types (INT, CHAR, {anything that a struct might be called})
vartype: CHAR
	| INT
	| UNDER
	| ALPHA
	| SAU;

//Called after each var definition, so it may be an array
//num could be POS_NUM, but I kept it that was so isGTZ returns a better error report if needed
arr: %empty
	| OB num CB arr { isGTZ($2); };


num: POS_NUM
	| SUB POS_NUM { $$ = -$2; };
```

```
num_oper: ADD
        | SUB
        | MUL
        | DIV
        | POW;


/************************************************** STRUCT
*******************************************************/

struct_wrapper: %empty
        | STRUCT struct ESTRUCT wrapper struct_wrapper
        | TYPEDEF STRUCT struct ESTRUCT vname //choose this position for struct name, since if it
is before ESTRCT token
        {                                                                       // there is
are shift/reduce conflicts, since varable names have
                                                                                // the same
tokens as the struct name!!!
//Σκεψου δηλαδή: Όταν δει ; και έπειτα προαιρετικά αλλαγη γραμμής, δεν ξέρει αν το επόμενο token
είναι vname ή struct name
                addStruct_name($5);
        }
        wrapper struct_wrapper;

struct: vname
        {
                addStruct_name($1); //mid-rule in order to add the struct name to the array and be
able to use it as a valid
        }                                       //variable type within itself
        wrapper VARS struct_vars;

struct_vars: vartype vname arr append_struct_vars { isVartype($1); }
        | wrapper vartype vname arr append_struct_vars { isVartype($2); };

append_struct_vars: SEMIC wrapper
        | SEMIC struct_vars
        | COMMA vname arr append_struct_vars;

/************************************************** STRUCT
*******************************************************/


/************************************************** FUNCTION
*******************************************************/

func: %empty
        | func FUNC fname
        {
                if(is_function($3,"")){
                        yyerror("Function already declaired!");
                }
                arr_of_scopes[scope_cnt].argc = 0;
```

```
                strcpy(cur_scope, $3); //TODO also save the amount of arguments!!
                strcpy(arr_of_scopes[scope_cnt].scope, $3);
                scope_cnt++;
        }
        OP func_param CP wrapper func_body RET return_type wrapper ENDFUNC wrapper;

main: SMAIN
        {
                strcpy(cur_scope, "main");
                strcpy(arr_of_scopes[scope_cnt].scope, "main");
                scope_cnt++;
        }
         wrapper func_body EMAIN end_wrapper;

func_param: %empty
        | vartype vname
        {
                add_type_of_arg($1);
                arr_of_scopes[scope_cnt-1].argc++;
                add_in_cur_scope($2);
        }
        arr append_func_param { isVartype($1); };

append_func_param: %empty
        | COMMA func_param;

//Increment the global counter of arguments,
//so you may then check if function call has the same amount of parameters
func_arg: %empty
        | ALPHA {arg_cnter++; is_declaired($1, "Variable not declaired in this scope!");} arr
append_func_arg
        | num {arg_cnter++;} append_func_arg
        | UNDER {arg_cnter++; is_declaired($1, "Variable not declaired in this scope!");} arr
append_func_arg
        | SAU {arg_cnter++; is_declaired($1, "Variable not declaired in this scope!");} arr
append_func_arg;

append_func_arg: %empty
        | COMMA func_arg;

func_body: VARS wrapper func_var
        | commands;

func_var: vartype vname
        {
                if(is_declaired($2, "")){
                        yyerror("There is already a variable with that name!");
                }
                add_in_cur_scope($2);
        }
        arr     append_func_vars { isVartype($1); };
```

```
append_func_vars: SEMIC func_var
        | SEMIC wrapper func_var
        | SEMIC commands
        | COMMA vname arr append_func_vars;

return_type: vname { is_declaired($1, "Variable is not declaired in this scope!"); }
        | num
        | BOOL;

/************************************************** FUNCTION
*******************************************************/


/************************************************** COMMANDS
*******************************************************/

commands: %empty
        | wrapper commands
        | assign commands
        | loop commands
        | flow_control commands
        | print commands;

extend_commands: %empty
        | wrapper extend_commands
        | assign extend_commands
        | loop extend_commands
        | flow_control extend_commands
        | print extend_commands;
        | BREAK SEMIC extend_commands;
                                                                        /************ ASSIGN
************/
assign: vname
        {
                is_declaired($1, "Variable is not declaired in this scope!");
        }
        EQS rvalue SEMIC;

rvalue: vname
        {
                is_declaired($1, "Variable is not declaired in this scope!");
        }
        append_rvalue
        | num append_rvalue
        | OP rvalue CP append_rvalue
        | fname
        {
                is_function($1,"Function is not declaired!");
                arg_cnter = 0;
        }
        OP func_arg CP
        {
```

```
                    is_argument_count_correct($1);
          }
          append_rvalue;

append_rvalue: %empty
          | num_oper rvalue;
                                                                                    /************ ASSIGN
************/

                                                                                    /************* LOOP
*************/
loop: while
          | for;

while: WHILE OP cond CP wrapper extend_commands EWHILE;

cond: rvalue
          | rvalue COMP_OPER cond
          | rvalue LOG_OPER cond;

for: FOR vname COLON_EQ num TO num STEP num wrapper extend_commands EFOR;

                                                                                    /************* LOOP
*************/

                                                                                    /************* FLOW
*************/
flow_control: if
          | switch;

if: IF cond THEN wrapper extend_commands else ENDIF;

else: %empty
          | ELSEIF wrapper extend_commands else
          | ELSE wrapper extend_commands;

switch: SWITCH OP rvalue CP wrapper commands case;

case: CASE OP rvalue CP COLON extend_commands case
          | DEFAULT COLON extend_commands ESWITCH
          | ESWITCH;
                                                                                    /************* FLOW
*************/

                                                                                    /************* OTHER
************/
print: PRINT OP STR print_var CP SEMIC;

print_var: %empty
          | COMMA vname print_var;
                                                                                    /************* OTHER
************/
```

```
/************************************************** COMMANDS
*********************************************************/


%%

//code from http://aquamentus.com/tut_lexyacc.html
// stuff from lex that bison needs to know about:
//count lines, so you may report on error
extern int line;
extern int yylex();
extern int yyparse();
extern FILE *yyin;

int main(int argc, char **argv){
	int i;

	// open a file handle to a particular file:
	if(argc != 2){
		printf("First argument must be the code file\n");
		exit(1);
	}

	FILE *codefile = fopen(argv[1], "r");
	if (!codefile) {
		printf("%s did not open!\n",argv[2]);
		exit(1);
	}

	//set lex to read from it instead of defaulting to STDIN:
	yyin = codefile;

	//parse through the input file until there is no more:
	do {
		yyparse();
	} while (!feof(yyin));

#if DEBUG
	printf("Scope: %s\n", cur_scope);
	for (int i = 0; i < scope_cnt; i++){
		printf("arr_of_scopes[%d] = %s\n", i, arr_of_scopes[i].scope);
		int j = 0;
		while(strcmp(arr_of_scopes[i].vars[j],"")){
			printf("\t %s,", arr_of_scopes[i].vars[j]);
			j++;
		}
		printf("\n\t %d arguments with types: \n", arr_of_scopes[i].argc);
		for(j = 0; j < arr_of_scopes[i].argc; j++){
			printf("\t %s\n", arr_of_scopes[i].argt[j]);
		}
	}
```

```
	for (int i = 0; i < MAX_STRUCTS; i++){
		if(!strcmp(struct_names[i],"")){
			break;
		}
		printf("struct_names[%d] = %s\n",i,struct_names[i]);
	}
#endif
}

void yyerror(const char *s){ //In OReilly chapter one there is no const
   fprintf(stderr, "error in line %d: %s\n", line, s);
	exit(1);
}

//TODO: when a negative val (ex -2) is given as the size of an array,
//the error created is because of the -, as a NUM_OPER token
//a fix could be extending the NUM declaration, so it also recognizes negative numbers
//Keep in mind that in flex NUM should be before NUM_OPERATOR, in order for this to work
void isGTZ(int i){
	if(i <= 0){
		yyerror("Array size must be greater than 0\n");
	}
}

//updates the array of structs, so the user may later define a var of type struct
void addStruct_name(char* str){
	int i;
	strcpy(struct_names[pos],str);
	pos++;
}

//Checks if variable type is valid, either INTEGER or CHAR tokens or a struct that has been previously
defined
int isVartype(char* str){
	int i;
	if(!strcmp(str, "INTEGER") || !strcmp(str, "CHAR")){
		return 1;
	}
	for (i = 0; i < MAX_STRUCTS; i++){
		if(!strcmp(str, "")){
			break; //avoid a few extra iterations
		}
		else if(!strcmp(str, struct_names[i])){
			return 1;
		}
	}
	char *msg = strcat(str," not a var type!\n");
	yyerror(msg);
	return 0;
}
```

```
//Adds var in the struct (named scope),
//that is in (scope_cnt-1) position of the arr_of_scopes,
//in field (named vars)
void add_in_cur_scope(char* var){
	int i=0;
	while(strcmp(arr_of_scopes[scope_cnt-1].vars[i],"")){ //strcmp: if match returns 0
		i++;
		if(i==MAX_NUM_of_VARS){
			yyerror("Exceeded the number of allowed variables!\n");
		}
	}
	strcpy(arr_of_scopes[scope_cnt-1].vars[i], var);
}

//Adds type in the struct (named scope),
//that is in (scope_cnt-1) position of the arr_of_scopes,
//in field (argt)
//I may use this later in order to check if a function call is correct
//(has the correct arguments)
//Or even if it has been previously defined
void add_type_of_arg(char* type){
	int i = 0;
	while(strcmp(arr_of_scopes[scope_cnt-1].vars[i],"")){ //strcmp: if match returns 0
		i++;
		if(i==MAX_ARGS){
			yyerror("Exceeded the number of allowed arguments!\n");
		}
	}
	strcpy(arr_of_scopes[scope_cnt-1].argt[i], type);
}

//Check if var is in the cur scope (thus it is declaired)
//Second argument = "" will not display an error message
int is_declaired(char* var, char* error_msg){
	int i = 0;
	while (strcmp(arr_of_scopes[scope_cnt-1].vars[i], "")){
		if(!strcmp(arr_of_scopes[scope_cnt-1].vars[i], var)){
			return 1;
		}
		i++;
	}
	if(strcmp(error_msg,"")){
		yyerror(error_msg);
	}
	return 0;
}

//Check if function is declaired
//Second argument = "" will not display an error message
int is_function(char* func, char* error_msg){
	int i = 0;
	while (strcmp(arr_of_scopes[i].scope, "")){
```

```
		if(!strcmp(arr_of_scopes[i].scope, func)){
			return 1;
		}
		i++;
	}
	if(strcmp(error_msg,"")){
		yyerror(error_msg);
	}
	return 0;
}

int is_argument_count_correct(char* func_name){
	int i;
	while(strcmp(func_name, arr_of_scopes[i].scope)){
		i++;
	}
	if(arr_of_scopes[i].argc == arg_cnter){
		return 1;
	}
	else if(arg_cnter > arr_of_scopes[i].argc){
		yyerror("Too many arguments!");
	}
	else{
		yyerror("Not enough arguments!");
	}
}
```