
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

---

# *ΕΡΓΑΣΤΗΡΙΟ ΜΕΤΑΓΛΩΤΤΙΣΤΩΝ*

## *ΟΜΑΔΑ ΕΡΓΑΣΤΗΡΙΟΥ: Β3 1)*

## *ΜΕΡΟΣ Β:*

*Ανάπτυξη κώδικα bison για δημιουργία ανεξάρτητου ΣΑ,*

*Σύνδεση κώδικα flex με κώδικα bison,*

*Διαχείριση λεκτικών και συντακτικών προειδοποιητικών*

*λαθών*

## *ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2022-2023*

---

**Σιδηρόπουλος Νικόλαος (ΠΑΔΑ-20390283)**

**Φράγκος Νικόλαος (ΠΑΔΑ-20390256)**

**Γουρδομιχάλης Δημήτριος (ΠΑΔΑ-20390043)**

**Τσελάνι Μαρίνο (ΠΑΔΑ-20390241)**

# ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

# Εισαγωγή

Στην εργασία αυτή αναλύουμε τη δημιουργία συντακτικών αναλυτών με τη χρήση των εργαλείων Bison και Flex. Το Bison είναι ένα εργαλείο που περιγράφει τη γραμματική μιας γλώσσας και δημιουργεί έναν αναλυτή που επεξεργάζεται τον πηγαίο κώδικα σύμφωνα με αυτήν τη γραμματική. Από την άλλη πλευρά, το Flex προσδιορίζει τα λεκτικά στοιχεία του προγράμματος και επιστρέφει τα αντίστοιχα tokens στον αναλυτή του Bison. Με τη συνεργασία των δύο εργαλείων, μπορούμε να δημιουργήσουμε έναν πλήρη συντακτικό αναλυτή για μια γλώσσα προγραμματισμού, όπως είναι η uni-python.

Αρχικά, χρησιμοποιήσαμε ως πρότυπο το αρχείο simple-bison-code.y που βρίσκεται στο eclass στο συμπιεσμένο φάκελο simple-bison-code.zip. Αντικαταστήσαμε τα πεδία "FILL ME", με κώδικα που υλοποιεί Συντακτική Ανάλυση πηγαίου κώδικα της γλώσσας Uni-Python. Δημιουργήσαμε επίσης ένα αρχείο input.txt, το οποίο περιέχει σωστές αλλά και λανθασμένες εκφράσεις της συγκεκριμένης γλώσσας. Τέλος, ως αποτέλεσμα της Συντακτικής Ανάλυσης, όταν μία έκφραση αναγνωρίζεται, εμφανίζεται αριθμημένη στο αρχείο εξόδου(output.txt).
Περεταίρω, προσαρμόσαμε τον κώδικα bison ώστε ο συντακτικός σας αναλυτής να μπορεί να συνεργαστεί με ρουτίνα λεκτικής ανάλυσης που θα παραχθεί μέσω του flex, το οποίο αποτελεί μια προσαρμογή του κώδικα flex που αναπτύχθηκε στο Α-3 μέρος της εργασίας, έτσι ώστε να δημιουργείται ρουτίνα λεκτικής ανάλυσης για τον συντακτικό αναλυτή που θα προκύψει από την προσαρμογή του κώδικα flex.

Στη συνέχεια, για τη διαχείριση λεκτικών και συντακτικών λαθών προσθέσαμε κώδικα που αφορά τους κανόνες της γραμματικής. Χρησιμοποιήθηκε η μέθοδος του πανικού.
Αναλυτικότερα, στο αρχείο simple-code-flex.l προσθέσαμε 4 ορισμούς, που αφορούν τα λάθη. Επίσης, υλοποιήσαμε τις συναρτήσεις yyerror() και prn(). Ενώ, στο αρχείο simple-bison-code.y, δημιουργήσαμε τους καταμετρητές errors και par_warn, που αποτελούν καταμετρητές λαθών και προειδοποιήσεων αντίστοιχα. Όταν αντιληφθεί ο Συντακτικός Αναλυτής κάποιο λάθος, αυξάνει τον αντίστοιχο καταμετρητή, και εμφανίζει σχετικό μήνυμα λάθους. Έπειτα, συνεχίζεται κανονικά η αναγνώριση της επόμενης συμβολοσειράς.

Τέλος, αναπτύξαμε ένα Makefile το οποίο αυτοματοποιεί την μεταγλώττιση και εκτέλεση του κώδικά.

## Αρχεία Makefile , Flex & Bison

### Αρχείο Flex (simple-flex-code.l):

```
%option  noyywrap


%{


#include <stdio.h>

#include <string.h>

#include <stdlib.h>


/* Header file pou periexei lista me ola ta tokens */

#include "simple-bison-code.tab.h"


/* Orismos metrhth trexousas grammhs */


void prn(char *s);


extern char *yytext;

int lex_warn=0;


/* Στο παρακάτω τμήμα του κώδικα, οριζούμε τα tokens που θα χρησιμοποιηθούν και αναφέρονται στο simple-bison-code.tab.h αρχείο, με την αντίστοιχη Κανονική Εκφραση.

  Τέλος,τροποποιήσαμε και μερικές απο τις κανονικές εκφράσεις που είχαμε βάλει στο Α μέρος της εργασίας, καθώς παρουσιάστηκαν σφάλματα κατα την διάρκεια της εκτέλεσης του simple-bison κώδικα.

*/


%}


DELIMITER       [ \t]+
```

DECIMAL ([0]|([1-9][0-9]*))

BINARY [0][b|B][01]+

FLOAT (([1-9][0-9]*)|0)(\.[0-9]+)?([eE][-+]?[0-9]+)?

IMAGINARY [+-]?(0(\.[0-9]+)+|[1-9][0-9]*(\.[0-9]+)?)([eE][+-]?[0-9]+)?[j]

IDENTIFIER [a-zA-Z_][a-zA-Z0-9_]*

STRING (\"(\\[\\\'"nr]|[^\n\'"\\])*\")|(\'(\\[\\\'"nr]|[^\n\'"\\])*\')

COMMENT #.*

KEYWORD
False|class|finally|is|return|None|continue|for|lambda|try|True|def|from|nonlocal|while|and|del|global|not|with|as|elif|if|or|yield|assert|else|import|pass|break|except|in|raise

S_WORD \(|\)|\[|\]|\{|\}|,|:|\.|;|@|=|->|\+=|\-=|*=|\/=|\/\/=|%=|@=|&=|\|=|\^=|>>=|<<=|**=

UNKNOWN_TOKEN .

/* Στην συνέχεια, ακολουθεί η περιοχή ορισμού των κανόνων που θα υπάρχουν κατα την μεταγλώττιση , δήλαδή τι ενέργειες θα πραγματοποιηθούν με την ανίχνευση περιεχομένου του αρχείου. */

/* Την περίπτωση των operator την χωρίσαμε για κάθε operator ξεχωριστά ενώ προσθέσαμε τις λέξεις print,len και το TOKEN_ERROR για αναγνώριση λαθών */

%x error

%%

"+" { return PLUS; }

"*" { return TIMES; }

"/" { return DIVIDE; }

"^" { return POWER; }

"%" { return MOD; }

">" { return GREATER; }

"<" { return LESS; }

">=" { return GREATER_EQ; }

"<=" { return LESS_EQ; }

"==" { return EQUAL; }

"!=" { return NOT_EQUAL; }

```
"("     { return LEFT_PAREN; }
")"     { return RIGHT_PAREN; }
"["             { return LS_BRACKET; }
"]"             { return RS_BRACKET; }
"{"             { return L_BRACKET; }
"}"             { return R_BRACKET; }
","             { return COMMA; }
":"             { return COLON; }
"."             { return DOT; }
";"             { return SEMICOLON; }
"="             { return ASSIGN; }
"+="    { return ADD_ASSIGN; }
"-="    { return SUB_ASSIGN; }
"*="    { return MULT_ASSIGN; }
"/="    { return DIV_ASSIGN; }

"print" { return PRINT; }
"len"   { return LEN; }


{DELIMITER}     {  }
{KEYWORD}               { yylval.sval = strdup(yytext); return KEYWORD; }
{S_WORD}        { return S_WORD; }
{DECIMAL}       { yylval.ival = atoi(yytext); return DECIMAL; }
{BINARY}        { yylval.sval = strdup(yytext); return BINARY; }
{FLOAT}                 { yylval.fval = atof(yytext); return FLOAT; }
{IMAGINARY}             { yylval.sval = strdup(yytext); return IMAGINARY; }
{IDENTIFIER}    { yylval.sval = strdup(yytext); return IDENTIFIER; }
{STRING}        { yylval.sval = strdup(yytext); return STRING; }
{COMMENT}               { return COMMENT; }
```

```
"-"                     { return MINUS; }

\n              { return NEWLINE; }

.               { lex_warn++; BEGIN(error); return TOKEN_ERROR; }
<error>[ \t]+   { BEGIN(0); printf("\n\t\t%d character(s) ignored so far\n",lex_warn); }
<error>.        { lex_warn++; }
<error>\n       { BEGIN(0); printf("\n\t\t%d character(s) ignored so far\n",lex_warn); return
NEWLINE; }



%%

void prn(char *s){
   printf("\tFlex->Matched token:%s\n",yytext);
}
```

**Αρχείο Bison (simple-bison-code.y):**

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

extern FILE *yyin;
extern FILE *yyout;

int yylex(void);
void yyerror(const char*);

/* Μετρητές σωστών και λανθασμένων εκφράσεων και λέξεων */
```

```
static int errors=0;
static int par_warn = 0;
static int correct = 0;
static int correct_words = 0;

#define YYDEBUG 1

/* Συναρτήσεις για μετατροπή από int,float σε string */
char* int_to_string(int num) {
    char* str = malloc(12 * sizeof(char));
    sprintf(str, "%d", num);
    return str;
}

char* float_to_string(float num) {
    char* str = malloc(20 * sizeof(char));
    sprintf(str, "%.2f", num);
    return str;
}

/* Συνάρτηση για επιστρογή μεγέθους λίστας-πλειάδας */
int getListSize(const char* string) {
    char* copy = strdup(string);
    int count = 0;
    char* token = strtok(copy, ",");
    while (token != NULL) {
        count++;
        token = strtok(NULL, ",");
    }
    free(copy);
    return count;
}
```

```
/* Συνάρτηση για συνένωση λίστας-πλειάδας */
char* concatWithComma(const char* str1, const char* str2) {
    size_t len1 = strlen(str1);
    size_t len2 = strlen(str2);
    size_t commaLen = strlen(", ");
    char* result = (char*)malloc((len1 + len2 + commaLen + 1) * sizeof(char));
    if (result == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        exit(1);
    }
    strcpy(result, str1);
    strcat(result, ", ");
    strcat(result, str2);
    return result;
}
%}

/* Δήλωση union για τον ορισμό διαφορετικών τύπων δεδομένων για τα αντίστοιχα tokens */
%union {
  int     ival;
  float fval;
  char   *sval;
}

/* Ορισμός των αναγνωριστικών λεκτικών μονάδων */
%token <ival> DECIMAL LEN
%token <fval> FLOAT
%token <sval> STRING BINARY IMAGINARY IDENTIFIER KEYWORD PRINT

%token S_WORD UNKNOWN_TOKEN COMMENT
%token PLUS MINUS TIMES DIVIDE POWER MOD GREATER LESS GREATER_EQ LESS_EQ
EQUAL NOT_EQUAL
```

```
%token LEFT_PAREN RIGHT_PAREN LS_BRACKET RS_BRACKET L_BRACKET R_BRACKET
COMMA COLON DOT SEMICOLON ASSIGN ADD_ASSIGN SUB_ASSIGN MULT_ASSIGN
DIV_ASSIGN

%token TOKEN_ERROR NEWLINE


/* Ορισμός προτεραιοτήτων των tokens */

%left   ASSIGN

%left   PLUS MINUS

%left   TIMES DIVIDE MOD

%left   NEG

%right POWER

%left   GREATER LESS GREATER_EQ LESS_EQ EQUAL NOT_EQUAL

%left   LS_BRACKET RS_BRACKET L_BRACKET R_BRACKET

%left   LEFT_PAREN RIGHT_PAREN


%type<ival> dec_expr

%type<fval> float_expr

%type<sval> string_expr

%type<sval> list_expr list_types


%%


/* Ορισμός των γραμματικών κανόνων, εκτέλεση του ανάλογου κώδικα και αύξηση των κατάλληλων
μετρητών */

program:

                | program expr;

      ;

expr:

                dec_expr NEWLINE                        { fprintf(yyout,"%d\n", $1); correct++; }

                | float_expr NEWLINE            { fprintf(yyout,"%.2f\n", $1); correct++; }

      | string_expr NEWLINE             { fprintf(yyout,"%s\n", $1); correct++; }

                | list_expr  NEWLINE            { fprintf(yyout,"%s\n", $1); correct++; }

                | COMMENT NEWLINE                       { }

                | NEWLINE                                       { }
```

```
                | TOKEN_ERROR                                 { errors++; }

                ;

dec_expr:

    DECIMAL                                                 { $$ = $1; correct_words++; }

    | dec_expr PLUS dec_expr                        { $$ = $1 + $3; correct_words++; }

                | dec_expr MINUS dec_expr                       { $$ = $1 - $3;
correct_words++; }

                | dec_expr TIMES dec_expr                       { $$ = $1 * $3;
correct_words++; }

                | dec_expr DIVIDE dec_expr                      { $$ = $1 / $3; correct_words++;
}

                | dec_expr MOD dec_expr                         { $$ = $1 % $3; correct_words++; }

                | dec_expr POWER dec_expr               { $$ = pow($1, $3); correct_words++; }

                | dec_expr GREATER dec_expr             { $$ = ($1 > $3) ? 1 : 0; correct_words++; }

                | dec_expr LESS dec_expr                { $$ = ($1 < $3) ? 1 : 0; correct_words++; }

                | dec_expr GREATER_EQ dec_expr          { $$ = ($1 >= $3) ? 1 : 0; correct_words++; }

                | dec_expr LESS_EQ dec_expr             { $$ = ($1 <= $3) ? 1 : 0; correct_words++; }

                | dec_expr EQUAL dec_expr               { $$ = ($1 == $3) ? 1 : 0; correct_words++; }

                | dec_expr NOT_EQUAL dec_expr           { $$ = ($1 != $3) ? 1 : 0; correct_words++; }

                | LEFT_PAREN dec_expr RIGHT_PAREN                       { $$ = $2;
correct_words+=2; }

                | MINUS dec_expr %prec NEG                              { $$ = -$2;
correct_words++; }

                | LEN LEFT_PAREN list_expr RIGHT_PAREN          { $$ = getListSize($3);
correct_words+=3; }


                /* Λάθος περιπτώσεις */

                |DECIMAL DECIMAL
        {printf("\tWarning!Integer starting with 0!\n");par_warn++; $$=$2;}

                |DECIMAL IDENTIFIER
        {printf("\tWarning!Identifier starting with number!\n");par_warn++; }

                |DECIMAL DOT
                {printf("\tWarning!Incomplete number!\n");par_warn++; }

                |DOT DECIMAL
                {printf("\tWarning!Incomplete number!\n");par_warn++; }

                |dec_expr TOKEN_ERROR PLUS dec_expr
{printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$=$1+$4;}

                |dec_expr TOKEN_ERROR MINUS dec_expr
```

```
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 - $4;}

                |dec_expr TOKEN_ERROR TIMES dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 * $4;}

                |dec_expr TOKEN_ERROR DIVIDE dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++;$$ = $1 / $4;}

                |dec_expr TOKEN_ERROR MOD dec_expr
{printf("\tWarning!Unknown operator ignored!\n");par_warn++;$$ = $1 % $4;}

                |dec_expr TOKEN_ERROR POWER dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = pow($1, $4);}

      |dec_expr TOKEN_ERROR GREATER dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = ($1 > $4) ? 1 : 0;}

      |dec_expr TOKEN_ERROR LESS dec_expr                {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 < $4) ? 1 : 0;}

      |dec_expr TOKEN_ERROR GREATER_EQ dec_expr          {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 >= $4) ? 1 : 0;}

      |dec_expr TOKEN_ERROR LESS_EQ dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++;$$ = ($1 <= $4) ? 1 : 0; }

      |dec_expr TOKEN_ERROR EQUAL dec_expr               {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 == $4) ? 1 : 0;}

      |dec_expr TOKEN_ERROR NOT_EQUAL dec_expr           {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 != $4) ? 1 : 0;}



      |dec_expr PLUS TOKEN_ERROR dec_expr                {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$=$1+$4;}

                |dec_expr MINUS TOKEN_ERROR dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 - $4;}

                |dec_expr TIMES TOKEN_ERROR  dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 * $4;}

                |dec_expr DIVIDE TOKEN_ERROR dec_expr
{printf("\tWarning!Unknown operator ignored!\n");par_warn++;$$ = $1 / $4;}

                |dec_expr MOD TOKEN_ERROR  dec_expr
{printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 % $4;}

                |dec_expr POWER TOKEN_ERROR  dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = pow($1, $4);}

      |dec_expr GREATER TOKEN_ERROR  dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = ($1 > $4) ? 1 : 0;}

      |dec_expr LESS TOKEN_ERROR  dec_expr               {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 < $4) ? 1 : 0;}

      |dec_expr GREATER_EQ TOKEN_ERROR  dec_expr     {printf("\tWarning!Unknown operator
ignored!\n");par_warn++; $$ = ($1 >= $4) ? 1 : 0;}

      |dec_expr LESS_EQ TOKEN_ERROR  dec_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++;$$ = ($1 <= $4) ? 1 : 0; }
```

```
      |dec_expr EQUAL TOKEN_ERROR  dec_expr
         {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = ($1 == $4) ? 1 : 0;}

      |dec_expr NOT_EQUAL TOKEN_ERROR  dec_expr       {printf("\tWarning!Unknown operator
ignored!\n");par_warn++; $$ = ($1 != $4) ? 1 : 0;}



      |dec_expr PLUS PLUS dec_expr                {printf("\tWarning:Consecutive use of
operator!\n");par_warn++; $$=$1+$4;}

                  |dec_expr TIMES TIMES dec_expr
{printf("\tWarning!Consecutive use of operator!!\n");par_warn++; $$ = $1 * $4;}

                  |dec_expr DIVIDE DIVIDE dec_expr
{printf("\tWarning!Consecutive use of operator!!\n");par_warn++;$$ = $1 / $4;}

                  |dec_expr MOD MOD dec_expr
{printf("\tWarning!Consecutive use of operator!\n");par_warn++;$$ = $1 % $4;}

                  |dec_expr POWER POWER dec_expr
{printf("\tWarning!Consecutive use of operator!!\n");par_warn++; $$ = pow($1, $4);}

      |dec_expr GREATER GREATER dec_expr                  {printf("\tWarning!Consecutive
use of operator!\n");par_warn++; $$ = ($1 > $4) ? 1 : 0;}

      |dec_expr LESS LESS dec_expr                  {printf("\tWarning!Consecutive use of
operator!\n");par_warn++; $$ = ($1 < $4) ? 1 : 0;}

      |dec_expr GREATER_EQ GREATER_EQ dec_expr          {printf("\tWarning!Consecutive
use of operator!\n");par_warn++; $$ = ($1 >= $4) ? 1 : 0;}

      |dec_expr LESS_EQ LESS_EQ dec_expr                  {printf("\tWarning!Consecutive
use of operator!\n");par_warn++;$$ = ($1 <= $4) ? 1 : 0; }

      |dec_expr EQUAL EQUAL dec_expr                  {printf("\tWarning!Consecutive use of
operator!\n");par_warn++; $$ = ($1 == $4) ? 1 : 0;}

      |dec_expr NOT_EQUAL NOT_EQUAL dec_expr                  {printf("\tWarning!Consecutive
use of operator!\n");par_warn++; $$ = ($1 != $4) ? 1 : 0;}

                  ;

float_expr:

      FLOAT                                                   { $$ = $1;
correct_words++; }

      | float_expr PLUS float_expr                { $$ = $1 + $3; correct_words++; }

                  | float_expr MINUS float_expr                  { $$ = $1 - $3; correct_words++;
}

                  | float_expr TIMES float_expr                  { $$ = $1 * $3; correct_words++;
}

                  | float_expr DIVIDE float_expr                  { $$ = $1 / $3; correct_words++;
}

                  | float_expr POWER float_expr          { $$ = pow($1, $3); correct_words++; }

                  | float_expr GREATER float_expr         { $$ = ($1 > $3) ? 1 : 0; correct_words++; }

                  | float_expr LESS float_expr          { $$ = ($1 < $3) ? 1 : 0; correct_words++; }
```

```
                | float_expr GREATER_EQ float_expr      { $$ = ($1 >= $3) ? 1 : 0; correct_words++; }

                | float_expr LESS_EQ float_expr        { $$ = ($1 <= $3) ? 1 : 0; correct_words++; }

                | float_expr EQUAL float_expr          { $$ = ($1 == $3) ? 1 : 0; correct_words++; }

                | float_expr NOT_EQUAL float_expr      { $$ = ($1 != $3) ? 1 : 0; correct_words++; }

                | LEFT_PAREN float_expr RIGHT_PAREN            { $$ = $2; correct_words+=2; }

                | MINUS float_expr %prec NEG                   { $$ = -$2; correct_words++; }


                /* Λάθος περιπτώσεις */

                |float_expr TOKEN_ERROR PLUS float_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$=$1+$4;}

                |float_expr TOKEN_ERROR MINUS float_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 - $4;}

                |float_expr TOKEN_ERROR TIMES float_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 * $4;}

                |float_expr TOKEN_ERROR DIVIDE float_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++;$$ = $1 / $4;}

                |float_expr TOKEN_ERROR POWER float_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = pow($1, $4);}

      |float_expr TOKEN_ERROR GREATER float_expr                {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 > $4) ? 1 : 0;}

      |float_expr TOKEN_ERROR LESS float_expr                   {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 < $4) ? 1 : 0;}

      |float_expr TOKEN_ERROR GREATER_EQ float_expr
         {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = ($1 >= $4) ? 1 : 0;}

      |float_expr TOKEN_ERROR LESS_EQ float_expr          {printf("\tWarning!Unknown operator
ignored!\n");par_warn++;$$ = ($1 <= $4) ? 1 : 0; }

      |float_expr TOKEN_ERROR EQUAL float_expr
         {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = ($1 == $4) ? 1 : 0;}

      |float_expr TOKEN_ERROR NOT_EQUAL float_expr              {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 != $4) ? 1 : 0;}


      |float_expr PLUS TOKEN_ERROR float_expr                   {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$=$1+$4;}

                |float_expr MINUS TOKEN_ERROR float_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 - $4;}

                |float_expr TIMES TOKEN_ERROR  float_expr
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = $1 * $4;}

                |float_expr DIVIDE TOKEN_ERROR float_expr
{printf("\tWarning!Unknown operator ignored!\n");par_warn++;$$ = $1 / $4;}

                |float_expr POWER TOKEN_ERROR  float_expr
```

```
        {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = pow($1, $4);}

    |float_expr GREATER TOKEN_ERROR  float_expr      {printf("\tWarning!Unknown operator
ignored!\n");par_warn++; $$ = ($1 > $4) ? 1 : 0;}

    |float_expr LESS TOKEN_ERROR  float_expr                {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 < $4) ? 1 : 0;}

    |float_expr GREATER_EQ TOKEN_ERROR  float_expr          {printf("\tWarning!Unknown
operator ignored!\n");par_warn++; $$ = ($1 >= $4) ? 1 : 0;}

    |float_expr LESS_EQ TOKEN_ERROR  float_expr         {printf("\tWarning!Unknown operator
ignored!\n");par_warn++;$$ = ($1 <= $4) ? 1 : 0; }

    |float_expr EQUAL TOKEN_ERROR  float_expr
       {printf("\tWarning!Unknown operator ignored!\n");par_warn++; $$ = ($1 == $4) ? 1 : 0;}

    |float_expr NOT_EQUAL TOKEN_ERROR  float_expr    {printf("\tWarning!Unknown operator
ignored!\n");par_warn++; $$ = ($1 != $4) ? 1 : 0;}



    |float_expr PLUS PLUS float_expr              {printf("\tWarning:Consecutive use of
operator!\n");par_warn++; $$=$1+$4;}

                |float_expr TIMES TIMES float_expr
{printf("\tWarning!Consecutive use of operator!!\n");par_warn++; $$ = $1 * $4;}

                |float_expr DIVIDE DIVIDE float_expr
{printf("\tWarning!Consecutive use of operator!!\n");par_warn++;$$ = $1 / $4;}

                |float_expr POWER POWER float_expr
{printf("\tWarning!Consecutive use of operator!!\n");par_warn++; $$ = pow($1, $4);}

    |float_expr GREATER GREATER float_expr                {printf("\tWarning!Consecutive
use of operator!\n");par_warn++; $$ = ($1 > $4) ? 1 : 0;}

    |float_expr LESS LESS float_expr                    {printf("\tWarning!Consecutive use of
operator!\n");par_warn++; $$ = ($1 < $4) ? 1 : 0;}

    |float_expr GREATER_EQ GREATER_EQ float_expr          {printf("\tWarning!Consecutive
use of operator!\n");par_warn++; $$ = ($1 >= $4) ? 1 : 0;}

    |float_expr LESS_EQ LESS_EQ float_expr                {printf("\tWarning!Consecutive
use of operator!\n");par_warn++;$$ = ($1 <= $4) ? 1 : 0; }

    |float_expr EQUAL EQUAL float_expr                    {printf("\tWarning!Consecutive
use of operator!\n");par_warn++; $$ = ($1 == $4) ? 1 : 0;}

    |float_expr NOT_EQUAL NOT_EQUAL float_expr            {printf("\tWarning!Consecutive
use of operator!\n");par_warn++; $$ = ($1 != $4) ? 1 : 0;}

                ;

string_expr:

                BINARY
                { $$ = $1; correct_words++; }

                | IMAGINARY
        { $$ = $1; correct_words++; }

                | IDENTIFIER                                                    { $$ =
```

```
$1; correct_words++; }

                | IDENTIFIER IDENTIFIER                                                  {
printf("\tWarning:Name with space!\n"); par_warn++; }

                | IDENTIFIER ASSIGN string_expr                          { $$ = $3;
correct_words++; }

                | IDENTIFIER ASSIGN dec_expr                             { $$ = int_to_string($3);
correct_words++; }

                | IDENTIFIER ASSIGN float_expr                                  { $$ =
float_to_string($3); correct_words++; }

                | STRING
        { $$ = $1; correct_words++; }

                | KEYWORD
        { $$ = $1; correct_words++; }

                | PRINT LEFT_PAREN STRING RIGHT_PAREN                { $$ = $3;
correct_words+=3; }

                | PRINT LEFT_PAREN DECIMAL RIGHT_PAREN               { $$ = int_to_string($3);
correct_words+=3; }

                | PRINT LEFT_PAREN FLOAT RIGHT_PAREN                 { $$ =
float_to_string($3); correct_words+=3; }

                | PRINT LEFT_PAREN IDENTIFIER RIGHT_PAREN    { $$ = $3; correct_words+=3;
}

                | PRINT LEFT_PAREN BINARY RIGHT_PAREN                { $$ = $3;
correct_words+=3; }

                | PRINT LEFT_PAREN IMAGINARY RIGHT_PAREN         { $$ = $3;
correct_words+=3; }

                | PRINT LEFT_PAREN LS_BRACKET list_types RS_BRACKET RIGHT_PAREN
        { $$ = $4; correct_words+=5; }

                | PRINT LEFT_PAREN LEFT_PAREN list_types RIGHT_PAREN RIGHT_PAREN
        { $$ = $4; correct_words+=5; }

                ;

list_expr:

                LS_BRACKET list_types RS_BRACKET                { $$ = $2; correct_words+=2; }

                | LEFT_PAREN list_types RIGHT_PAREN             { $$ = $2; correct_words+=2; }

                | LS_BRACKET list_types RS_BRACKET PLUS LS_BRACKET list_types
RS_BRACKET              { $$ = concatWithComma($2, $6); correct_words+=5; }

                | LEFT_PAREN list_types RIGHT_PAREN PLUS LEFT_PAREN list_types
RIGHT_PAREN                     { $$ = concatWithComma($2, $6); correct_words+=5; }

                ;

/* Χειρισμός λιστών και πλειάδων με string */

list_types:
```

```
STRING                          { $$ = strdup($1); correct_words++; }
| list_types COMMA STRING
{
        size_t list_len = strlen($1);
        size_t elem_len = strlen($3);
        $$ = malloc((list_len + 2 + elem_len) * sizeof(char));
        strcpy($$, $1);
        strcat($$, ", ");
        strcat($$, $3);
        free($1);
}
| DECIMAL                 { $$ = int_to_string($1); correct_words++; }
| list_types COMMA DECIMAL
{
        char* str = int_to_string($3);
        size_t list_len = strlen($1);
        size_t elem_len = strlen(str);
        $$ = malloc((list_len + elem_len + 2) * sizeof(char));
        strcpy($$, $1);
        strcat($$, ", ");
        strcat($$, str);
        free($1);
        free(str);
}
| FLOAT                         { $$ = float_to_string($1); correct_words++; }
| list_types COMMA FLOAT
{
        char* str = float_to_string($3);
        size_t list_len = strlen($1);
        size_t elem_len = strlen(str);
        $$ = malloc((list_len + elem_len + 2) * sizeof(char));
        strcpy($$, $1);
```

```
                    strcat($$, ", ");
                    strcat($$, str);
                    free($1);
                    free(str);
                }
                ;

%%

/* Διαχείρηση συντακτικών λαθών με την yyerror */
void yyerror(const char* message) {
    fprintf(stderr, "Error at line %d: %s\n", par_warn, message);
}


/* Συνάρτηση main που διαχειρίζεται τα αρχεία input.txt-output.txt */
int main(int argc,char **argv){
                //yydebug = 1;
        if(argc == 3){
            if(!(yyin = fopen(argv[1], "r"))) {
                fprintf(stderr, "Cannot read file: %s\n", argv[1]);
                    return 1;
            }
            if(!(yyout = fopen(argv[2], "w"))) {
                fprintf(stderr, "Cannot create file: %s\n", argv[2]);
                    return 1;
            }
        }
        else if(argc == 2){
            if(!(yyin = fopen(argv[1], "r"))) {
                fprintf(stderr, "Cannot read file: %s\n", argv[1]);
                    return 1;
```

```
            }

        }



        int ret=yyparse();


                    /* Παρουσίαση των λανθασμένων(errors/warnings) και σωστών λέξεων-εκφράσεων */

        if(ret==0){

            printf("\n\t\tBison->Parsing Succeeded.\n");

            fprintf(stderr,"Parsing completed with %d correct expression(s), %d correct words(s), %d error(s)
  and %d warning(s)\n",correct,correct_words,errors,par_warn);

                                fprintf(yyout,"\nParsing completed with : \n");

                                fprintf(yyout,"\nCorrect expression(s) : %d\n",correct);

                                fprintf(yyout,"\nCorrect word(s) : %d\n",correct_words);

                                fprintf(yyout,"\nWarning(s) : %d\n",par_warn);

                                fprintf(yyout,"\nError(s) : %d\n",errors);

            printf("\n");

        }else{

            printf("\t\tBison->Parsing Failed");

                                fprintf(yyout,"\t\tBison->Parsing Failed");

        }


       return 0;

}
```

## Makefile:

```
all:
        bison -d simple-bison-code.y
        flex simple-flex-code.l
        gcc simple-bison-code.tab.c lex.yy.c -lm -lfl -o flex-bison
        ./flex-bison input.txt output.txt

clean:
        rm uni-c.tab.c uni-c.tab.h lex.yy.c

flex:
```

```
	flex -o simple-flex-code.c simple-flex-code.l
	gcc -o simple-flex-code simple-flex-code.c
	./simple-flex-code input.txt output.txt

clean-flex:
	rm simple-flex-code simple-flex-code.c

bison:
	bison -o simple-bison-code.c simple-bison-code.y
	gcc -o simple-bison-code simple-bison-code.c
	./simple-bison-code input.txt output.txt

clean-bison:
	rm simple-bison-code.c simple-bison-code
```

## Είσοδος και έξοδος

Αρχεία: input.txt & output.txt

Δημιουργήσαμε το αρχείο input.txt το οποίο περιλαμβάνει ένα σύνολο έγκυρων καιμη περιπτώσεων για την δοκιμή κάθε κατηγορίας token της γλώσσας uni-python.

### input.txt:

#Valid cases

#Operators

1+3+4

5-3

2*3

6/3

7%4

3^2

1 < 4

5 > 2

3 >= 2

4 <= 4

1 == 1

1 != 3

#Special words

(1 + 2)

(1,2,3,4)

[1,2.3,"ad"]

a = 5

#Integer

0

```
1
7
-10

#Binary
0b1111
0B0000
0b0101010010101010

#Float
1.2
1e10
1E10
0e0

#Imaginary
1j
2j
-3j
1.2E10j
1.5e-10j

#Indentifiers
Potteridis13
_test

#Strings
"test 1"
'test 2'
"Hello world!"
'Mark said, \"Boo!\"\n'
"Dimitris, Marinos, Nikos, Nikos"
```

```
#List-Tuples concat
(1,2) + (3,4)
[1,2.3,"ad"] + ['ad',7]

#Len
len ((1,2))
len ([1,2.3,"ad"])

#Print
print("test")

#Invalid cases

#Integer
1++2
01
10c

#Binary
0b
0B
1b1111
0B2

#Float
4.
.1
1e

#Imaginary
0j

#Indentifiers
```

8nikos

Mark Smith

#Strings

"test'

"test2

'test2"

'test3

Από το παραπάνω input αρχείο και τον κώδικα παράγεται το output.txt

## output.txt:

8

2

6

2

3

9

1

1

1

1

1

1

3

1, 2, 3, 4

1, 2.30, "ad"

5

0

1

7

-10

0b1111

0B0000

0b010101001010101

1.20

10000000000.00

10000000000.00

0.00

1j

2j

-3j

1.2E10j

1.5e-10j

Potteridis13

_test

"test 1"

'test 2'

"Hello world!"

'Mark said, \"Boo!\"\n'

"Dimitris, Marinos, Nikos, Nikos"

1, 2, 3, 4

1, 2.30, "ad", 'ad', 7

2

3

"test"

3

1

10

0

0

1

0

4

4

1

0

8

Mark

Parsing completed with :

Correct expression(s) : 57

Correct word(s) : 106

Warning(s) : 13

Error(s) : 4

# Γενικές παρατηρήσεις/Σχολιασμός αποτελεσμάτων :

Στο αρχείο output.txt εμφανίζονται τα αποτελέσματα των γραμματικών κανόνων που ορίζονται στον κώδικα του συντακτικού αναλυτή. Επίσης εμφανίζεται το πλήθος των γραμματικών λαθών (syntax errors και warnings) και των γραμματικά σωστών εκφράσεων/λέξεις. Σε περίπτωση που υπάρχει γραμματικό λάθος εμφανίζεται το κατάλληλο μήνυμα στο τερματικό.

## Αναφορά για διαδικασία μεταγλώττισης των αρχείων flex και bison και εκτέλεσης του τελικού παραγώμενου προγράμματος

Η μεταγλώττιση και η εκτέλεση του προγράμματος της συντακτικής ανάλυσης πραγματοποιήθηκε με επιτυχία και χωρίς την ανάγκη αντιμετώπισης προβλημάτων.

## Αρμοδιότητες μελών:

Για την εκπόνηση της συγκεκριμένης εργασίας υπήρξε δια ζώσης και εξ' αποστάσεως συνεργασία μεταξύ των μελών της ομάδας, που αναγράφονται παρακάτω. Αναλυτικότερα, τα εξής μέλη ασχολήθηκαν με το σύνολο των τμημάτων της εργασίας:

- **Σιδηρόπουλος Νικόλαος: Συγγραφή αρχείου εισόδου, συγγραφή κώδικα, σύνταξη τεκμηρίωσης, σχολιασμός κώδικα και αποτελεσμάτων.**
- **Φράγκος Νικόλαος: Συγγραφή αρχείου εισόδου, συγγραφή κώδικα, σύνταξη τεκμηρίωσης, σχολιασμός κώδικα και αποτελεσμάτων.**
- **Γουρδομιχάλης Δημήτριος: Συγγραφή αρχείου εισόδου, συγγραφή κώδικα, σύνταξη τεκμηρίωσης, σχολιασμός κώδικα και αποτελεσμάτων.**
- **Τσελάνι Μαρίνο: Συγγραφή αρχείου εισόδου, συγγραφή κώδικα, σύνταξη τεκμηρίωσης, σχολιασμός κώδικα και αποτελεσμάτων.**

- **Βούλγαρης Γιώργος: Αποχώρησε μετά το Α2 μέρος της εργασίας.**