ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΡΟΣ Α3

ΣΥΜΠΛΗΡΩΣΗ ΠΡΟΤΥΠΟΥ ΚΩΔΙΚΑ FLEX

ΣΤΟΙΧΕΙΑ ΟΜΑΔΑΣ / ΕΡΓΑΣΙΑΣ

ΑΡΙΘΜΟΣ ΟΜΑΔΑΣ: 2
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 1: Αθανασίου Βασίλειος Ευάγγελος (ΠΑΔΑ-19390005)
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 2: Θεοχάρης Γεώργιος (ΠΑΔΑ-19390283)
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 3: Τάτσης Παντελής (ΠΑΔΑ-20390226)
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 4: Ηλιού Ιωάννης (ΠΑΔΑ-19390066)
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 5: Δομινάρης Βασίλειος (ΠΑΔΑ-21390055)

ΤΜΗΜΑ ΕΡΓΑΣΤΗΡΙΟΥ: Β1 Τετάρτη 12:00-14:00
ΥΠΕΥΘΥΝΟΣ ΕΡΓΑΣΤΗΡΙΟΥ: Ιορδανάκης Μιχάλης
ΠΡΟΘΕΣΜΙΑ ΥΠΟΒΟΛΗΣ: 07/05/2024
ΗΜΕΡΟΜΗΝΙΑ ΟΛΟΚΛΗΡΩΣΗΣ: 07/05/2024

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

ΠΕΡΙΕΧΟΜΕΝΑ

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Εισαγωγή

Αναφορά

Το μέρος “Α3 Συμπλήρωση πρότυπου κώδικα FLEX” αποτελεί το 2ο μέρος της εργασίας “Δημιουργία Ανεξάρτητου Λεκτικού Αναλυτή με τη γεννήτρια FLEX”. Η αναφορά εστιάζει στην παρουσίαση του κώδικα FLEX που αναγνωρίζει τις λεκτικές μονάδες της γλώσσας Uni-C μέσα από τις κανονικές εκφράσεις που υλοποιήθηκαν από την ομάδα στο 1ο μέρος “Α2 Κωδικοποίηση αυτόματων πεπερασμένων καταστάσεων μέσω FSM”. Μέσα από εξαντλητικές δοκιμαστικές εκτελέσεις που πραγματοποιούνται στην παρούσα αναφορά, ξεκαθαρίζεται η σωστή λειτουργία του λεκτικού αναλυτή η οποία θα αποτελέσει τη βάση για την υλοποίηση της συντακτικής ανάλυσης στο Β μέρος.

Οργάνωση ομάδας

Μέσα από το κλίμα συνεργασίας, όλα τα μέλη της ομάδας συνέβαλλαν στην συμπλήρωση του κώδικα FLEX, ώστε να γίνει αντιληπτή η λειτουργία του και πώς ακριβώς συνδέεται με τις εργασίες που κάναμε στο 1ο μέρος της εργασίας με τα αυτόματα πεπερασμένα. Στη συνέχεια, οι εργασίες μοιράστηκαν όπως και στο 1ο μέρος σε υπο-ομάδες, εργασίες όπως είναι ο σχολιασμός του κώδικα, οι επιλογές των δοκιμών εισόδου, αλλά και ο εκτενής σχολιασμός των αποτελεσμάτων .

Περιεχόμενο αναφοράς

Η παρουσίαση εστιάζει στην λειτουργικότητα του λεκτικού αναλυτή με τη γεννήτρια FLEX, η οποία αναγνωρίζει τις λεκτικές μονάδες της γλώσσας Uni-C. Τα κεφάλαια αναφοράς είναι τα εξής:

1. Κώδικας FLEX
2. Περιπτώσεις ελέγχου
3. Σχολιασμός αποτελεσμάτων

Επιπρόσθετα, για κάθε εργασία αναγράφονται:

1. Ποια υπο-ομάδα την υλοποίησε
2. Σχολιασμός των περιπτώσεων ελέγχου
3. Σχόλια όσον αφορά τον τρόπο λειτουργίας της προσομοίωσης, τυχόν δυσκολίες που προέκυψαν και τρόποι αντιμετώπισης
4. Αναφορά τυχόν ελλείψεων και ορθότητας ως προς την εκτέλεση του κώδικα

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Ανάλυση αρμοδιοτήτων/ρόλων όλων των μελών της ομάδας

Αθανασίου Βασίλειος Ευάγγελος (ΠΑΔΑ-19390005)

Στον ρόλο του γενικού συντονιστή, ο φοιτητής συμμετείχε στις εργασίες:

- Εισαγωγή
- 1. Tokens
- 2. Κώδικας FLEX
- 3. Περιπτώσεις ελέγχου
- 4. Σχολιασμός αποτελεσμάτων
- Συγγραφή αναφοράς

Θεοχάρης Γεώργιος (ΠΑΔΑ-19390283)

Στον ρόλο του προγραμματιστή, ο φοιτητής συμμετείχε στις εργασίες:

- Εισαγωγή
- 1. Tokens
- 2. Κώδικας FLEX
- 3. Περιπτώσεις ελέγχου
- 4. Σχολιασμός αποτελεσμάτων

Τάτσης Παντελής (ΠΑΔΑ-20390226)

Στον ρόλο του δοκιμαστή (tester), ο φοιτητής συμμετείχε στις εργασίες:

- Εισαγωγή
- 1. Tokens
- 2. Κώδικας FLEX
- 3. Περιπτώσεις ελέγχου
- 4. Σχολιασμός αποτελεσμάτων

Ηλιού Ιωάννης (ΠΑΔΑ-19390066)

Στον ρόλο του σχολιαστή, ο φοιτητής συμμετείχε στις εργασίες:

- Εισαγωγή
- 1. Tokens
- 2. Κώδικας FLEX

- 3. Περιπτώσεις ελέγχου
- 4. Σχολιασμός αποτελεσμάτων

Δομινάρης Βασίλειος (ΠΑΔΑ-21390055)

Στον ρόλο του αναλυτή, ο φοιτητής συμμετείχε στις εργασίες:

- Εισαγωγή
- 1. Tokens
- 2. Κώδικας FLEX
- 3. Περιπτώσεις ελέγχου
- 4. Σχολιασμός αποτελεσμάτων

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

1. Tokens

Ο πίνακας με τα tokens βρίσκεται στο header αρχείο **token.h**.

```
#define DELIMITER 1

#define IDENTIFIER 2

#define STRING 3

#define INTEGER 4

#define FLOAT 5

#define KEYWORD 6

#define OPERATOR 7

#define NEWLINE 8

#define END_OF_FILE 9

#define UNKNOWN 10
```

2. Κώδικας FLEX

Ο κώδικας FLEX βρίσκεται στο πηγαίο αρχείο **simple-flex-code.l**

```
/* Όνομα αρχείου:       simple-flex-code.l

   Περιγραφή:            Υποδείγμα για ανάπτυξη λεκτικού αναλυτή με χρήση του εργαλείου
Flex

   Συγγραφέας:             Εργαστήριο Μεταγλωττιστών, Τμήμα Μηχανικών Πληροφορικής και
Υπολογιστών,

                          Πανεπιστήμιο Δυτικής Αττικής

   Σχόλια:               Το παρόν πρόγραμμα υλοποιεί (με τη χρήση flex) έναν απλό λεκτικό
αναλυτή

                          που αναγνωρίζει κενά (διάστημα και tab) και αριθμούς (δεκαδικού
συστήματος

                          μόνο!) για τη γλώσσα Uni-C ενώ διαχειρίζεται τους ειδικούς
χαρακτήρες

                          νέας γραμμής '\n' (new line) και 'EOF' (end of file). Υπάρχουν
αναφορές

                          για την αναγνώριση μεταβλητών, με τον πραγματικό κώδικα να έχει
αντικατασταθεί
```

```
                        από τον λεκτικό FILL ME έτσι ώστε να συμπληρωθεί από εσάς. Προαιρετικά ο λεκτικός
                        αναλυτής δέχεται ορίσματα αρχείων για είσοδο και έξοδο.
   Οδηγίες εκτέλεσης:     Δώστε "make" χωρίς τα εισαγωγικά στον τρέχοντα κατάλογο. Εναλλακτικά:
                        flex -o simple-flex-code.c simple-flex-code.l
                        gcc -o simple-flex-code simple-flex-code.c
                        ./simple-flex-code
*/


/* Η ανάγνωση περιορίζεται σε ένα μόνο αρχείο και τερματίζεται στο πρώτο EOF */
%option noyywrap



/* Κώδικας C για τον ορισμό των απαιτούμενων header files και των μεταβλητών.
   Οτιδήποτε ανάμεσα στα %{ και %} μεταφέρεται αυτόματα στο αρχείο C που
   θα δημιουργήσει το Flex. */

%{

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



/* Αρχείο κεφαλίδας που περιέχει λίστα με όλα τα tokens */
#include "token.h"

/* Ορισμός μετρητή τρέχουσας γραμμής */
int line = 1;

%}
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

```
/* Ονόματα και αντίστοιχοι ορισμοί (υπό μορφή κανονικής έκφρασης).

   Μετά από αυτό, μπορεί να γίνει χρήση των ονομάτων (αριστερά) αντί των,

   συνεπώς ιδιαίτερα μακροσκελών και δυσνόητων, κανονικών εκφράσεων */



DELIMITER        [;]

IDENTIFIER       [a-zA-Z_][a-zA-Z0-9_]{0,31}

STRING           \"([^"\\]*(\\[\\n"][^"\\]*)*)\"

INTEGER          ([1-9][0-9]*|0[x|X][0-9A-F]+|0[0-7]+|0)

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

KEYWORD
(break|case|const|continue|do|double|else|float|for|if|int|long|return|sizeof|struct|swi
tch|void|while|func)

OPERATOR                          (\+|-|\*|\/|%|=|\+=|-=-|\*=|\/=|!|&&|\|\||==|!=|\+\+|--
|<|>|<=|>=|&|\||\^|<<|>>|&)

COMMENT          (\/\/.*\n|\/\*[^*]*\*+([^/*][^*]*\*+)*\/)

WHITESPACE       [ \t]+

NEWLINE          [\n]

UNKNOWN          [^ \t\r\n;]+



/* Για κάθε πρότυπο (αριστερά) που ταιριάζει, εκτελείται ο αντίστοιχος

   κώδικας μέσα στα αγκίστρα. Η εντολή return επιτρέπει την επιστροφή

   μιας αριθμητικής τιμής μέσω της συνάρτησης yylex() */


%%


{DELIMITER}     { return DELIMITER; }

{IDENTIFIER}    { return IDENTIFIER; }

{STRING}        { return STRING; }

{INTEGER}       { return INTEGER; }

{FLOAT}         { return FLOAT; }
```

```
{KEYWORD}       { return KEYWORD; }
{OPERATOR}      { return OPERATOR; }
{COMMENT}       { ECHO; line++; }
{WHITESPACE}    { }
{NEWLINE}       { return NEWLINE; }
<<EOF>>         { return END_OF_FILE; }
{UNKNOWN}       { return UNKNOWN; }

%%

/* Πίνακας με όλα τα tokens αντίστοιχα με τους ορισμούς στο token.h */
char *tname[] = {
   "DELIMITER",
   "IDENTIFIER",
   "STRING",
   "INTEGER",
   "FLOAT",
   "KEYWORD",
   "OPERATOR",
   "NEWLINE",
   "END_OF_FILE",
   "UNKNOWN"
};


/* Η συνάρτηση main: Ο παρακάτω κώδικας θα τοποθετηθεί αυτόματα στο
   πρόγραμμα C που θα δημιουργήσει το Flex και θα αποτελέσει το αρχικό
   σημείο εκτέλεσης της εφαρμογής του λεκτικού αναλυτή. */

int main(int argc, char **argv){
        int token;
```

```
	/* Γίνεται ο έλεγχος των ορισμάτων της γραμμής εντολών. Αν τα
	   ορίσματα είναι 3, το πρόγραμμα διαβάζει από το αρχείο του 2ου
	   ορίσματος και γράφει στο αρχείο του 3ου. Αν τα ορίσματα είναι
	   2, διαβάζει από το αρχείο του 2ου ορίσματος και γράφει στην οθόνη.
	   Υποθέτεται ότι το 1ο όρισμα (argv[0]) στη C είναι το όνομα
	   του ίδιου του εκτελέσιμου αρχείου. */

	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;
		}
	}

	/* Η συνάρτηση yylex διαβάζει χαρακτήρες από την είσοδο και προσπαθεί
	   να αναγνωρίσει tokens. Τα tokens που αναγνωρίζει είναι αυτά που έχουν
	   οριστεί στο παρόν αρχείο, μεταξύ των %% και %%. Αν ο κώδικας που
	   αντιστοιχεί σε ένα πρότυπο περιέχει την εντολή 'return ΤΙΜΗ', η yylex()
	   επιστρέφει αυτή την τιμή και αποθηκεύει το αποτέλεσμα στη μεταβλητή token. */

	while( (token=yylex()) >= 0){
		/* Για κάθε αναγνωρισμένο token, εκτυπώνεται η γραμμή στην οποία βρέθηκε
```

```
                    και το όνομά του μαζί με την τιμή του. */
                if (token)
                {
                  switch(token)
                  {
                     case NEWLINE:
                        line++;
                        break;
                     case END_OF_FILE:
                        fprintf(yyout,  "\tLine=%d,  token=%s,  value=\"%s\"\n",  line,
tname[token-1], yytext);
                        printf("#END-OF-FILE#\n");
                        exit(0);
                        break;
                     case UNKNOWN:
                        fprintf(yyout,  "\tLine=%d,  token=%s  TOKEN,  value=\"%s\"\n",
line, tname[token-1], yytext);
                        break;
                     default:
                        fprintf(yyout,  "\tLine=%d,  token=%s,  value=\"%s\"\n",  line,
tname[token-1], yytext);
                        break;
                  }
                }


        }
        return 0;
}
```

3. Περιπτώσεις ελέγχου

Οι περιπτώσεις ελέγχου βρίσκονται στο text αρχείο **input.txt**.

```
// ====== Test case #1 IDENTIFIERS ======

a1

_1

Spectacular

a

A1

SPEctacular_15_

_a

_W

_ _A

a_b_c_1_2_3

AA4_F

1_id

/id

id.student

i d student

E2

_user

 user


// ====== Test case #2 STRINGS ======

"KALHMERA"

""

"hello, \"world!\"\n"

"\"\\n\"\"\\n\""

f"number of loops is {counter}"

"6/3=2"

"He said "Why Brutus?""

"6\3=2"
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

```
"6\\3=2"

"\n\"\\\d"

"Καλημέρα κόσμε"

'La vida loca'

"Hi Mark"

"Tests"

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

"Hi/n\n"


// ====== Test case #3 INTEGERS ======

0

3

214748

-50

0x4F

0X88AA

063

00

01

09

0XFGA9

01578

00xAFB1

-001

-0xAF01

0xff23

+01

+56

0x-FF


// ====== Test case #4 FLOATS ======

9e8
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

```
9E-8

3.14

10.0

10.0001

5e-15

1e100

3.1E0

0e0

-5.1e-100

0

-0.5

5.e1

1E1

00.01

0e01

5e1.5e1

1E1.2

6,20

0.0


// ====== Test case #5 COMMENTS ======

// hello world!!

/hello world!!/

# hello world!

// hello

hello world

/* hello world!! */

//hello//world//!!

\\ hello world!!

/*

 * hello

 * world
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

```
 * !!
 */
/*
hi
*/
\* hello world! *\
/*//
//hi
//* hi *//
%% hi
// hi */
/* hi //
// Αυτό είναι ένα σχόλιο
*/

// ====== Test case #6 KEYWORDS ======
int x = 5;
float y;
string str = "";
switch x;
case 1:
    break;
case 2:
    break;
sizeof(str);
sizeof;
double = 3.14;
const int z = 5;
struct Employee;
if x == 0
else x != 0
return 0;
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

```
// ====== Test case #7 FINAL ======
user;
a=(b+c);
"theo"
45
4.8
//hi
4.7
"good moring"
4e2
good

// ====== Test case #8 FINAL ======
"pao"
/*20*/
8.5
+
+---
break
func

// ====== Test case #9 FINAL ======
@#$
8,3
1_tzigger
/f
0.0
5.001

// ====== Test case #10 OPERATORS ======
x >= 5;
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

```
5 + 3 = 8;

6/=3;

x++;

x*y;

while 1;

void;

TRUE && FALSE == FALSE;

TRUE || TRUE == TRUE;

y--;

&bit


// ====== Test case #11 CODE ======

switch 1

    case 1

        break;

    case 2

        break;

return 0;


// ====== Test case #12 CODE ======

int x + y = z;

var x + y = z;

for i = 1; i <= N; i++ do

    if i % 2 == 0 then

        return 0;

    else

        return 1;

    end if;

end for;


// ====== Test case #13 CODE ======

const double pi=3.14;
```

```
struct Math m1;

m1.count = 3.14 * 2;

m1.length = 3.14 + 2 / 5;


// ====== Test case #14 CODE/COMMENTS ======

func calculate_perimeter_of_table void

    int length = 5;  // Length of table

    int width = 10;  // Width of table

    int x;


    x = length * 2 + width * 2; // Perimeter of table

    return x;
```

4. Σχολιασμός αποτελεσμάτων

Τα αποτελέσματα των περιπτώσεων ελέγχου βρίσκονται στο text αρχείο **output.txt**.

[Σχολιασμός]

Στη γραμμή 10 ο λεκτικός αναλυτής αγνοεί το whitespace χαρακτήρα (\s) με αποτέλεσμα να δέχεται ξεχωριστά τους χαρακτήρες “_”, “_A”. Ωστόσο και ξεχωριστά τους αναγνωρίζει σωστά ως identifiers. Στη γραμμή 16 συμβαίνει το ίδιο με το “i d student”, όπου ο αναλυτής δέχεται ξεχωριστά τα “i”, “d”, “student”.

```
// ====== Test case #1 IDENTIFIERS ======

      Line=2, token=IDENTIFIER, value="a1"


      Line=3, token=IDENTIFIER, value="_1"


      Line=4, token=IDENTIFIER, value="Spectacular"


      Line=5, token=IDENTIFIER, value="a"


      Line=6, token=IDENTIFIER, value="A1"


      Line=7, token=IDENTIFIER, value="SPEctacular_15_"
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

```
Line=8, token=IDENTIFIER, value="_a"

Line=9, token=IDENTIFIER, value="_W"

Line=10, token=IDENTIFIER, value="_"
Line=10, token=IDENTIFIER, value="_A"

Line=11, token=IDENTIFIER, value="a_b_c_1_2_3"

Line=12, token=IDENTIFIER, value="AA4_F"

Line=13, token=UNKNOWN TOKEN, value="1_id"

Line=14, token=UNKNOWN TOKEN, value="/id"

Line=15, token=UNKNOWN TOKEN, value="id.student"

Line=16, token=IDENTIFIER, value="i"
Line=16, token=IDENTIFIER, value="d"
Line=16, token=IDENTIFIER, value="student"

Line=17, token=UNKNOWN TOKEN, value="E2"

Line=18, token=IDENTIFIER, value="_user"

Line=19, token=IDENTIFIER, value="user"
```

[Σχολιασμός]

Στη γραμμή 26 ο λεκτικός αναλυτής αναγνωρίζει τα λάθη ως unknown tokens, αλλά χωρίζει τις σωστές εκφράσεις σε identifiers λόγω του whitespace (\s). Στη γραμμή 28 αναγνωρίζει το λανθασμένο string ως unknown token. Στη γραμμή 33 επίσης χωρίζει τις λέξεις "la", "vida", "loca" λόγω της ύπαρξης πάλι του whitespace (\s).

```
// ====== Test case #2 STRINGS ======
      Line=22, token=STRING, value=""KALHMERA""

      Line=23, token=STRING, value="""

      Line=24, token=STRING, value=""hello, \"world!\"\n""

      Line=25, token=STRING, value=""\"\\n\"\"\\n\""

      Line=26, token=UNKNOWN TOKEN, value="f"number"
      Line=26, token=IDENTIFIER, value="of"
      Line=26, token=IDENTIFIER, value="loops"
      Line=26, token=IDENTIFIER, value="is"
      Line=26, token=UNKNOWN TOKEN, value="{counter}""

      Line=27, token=STRING, value=""6/3=2""

      Line=28, token=STRING, value=""He said ""
      Line=28, token=IDENTIFIER, value="Why"
      Line=28, token=UNKNOWN TOKEN, value="Brutus?"""

      Line=29, token=UNKNOWN TOKEN, value=""6\3=2""

      Line=30, token=STRING, value=""6\\3=2""

      Line=31, token=UNKNOWN TOKEN, value=""\n\"\\\d""

      Line=32, token=STRING, value=""Καλημέρα κόσμε""
```

```
Line=33, token=UNKNOWN TOKEN, value="'La"
Line=33, token=IDENTIFIER, value="vida"
Line=33, token=UNKNOWN TOKEN, value="loca'"

Line=34, token=STRING, value=""Hi Mark""

Line=35, token=STRING, value=""Tests""

Line=36, token=STRING, value=""Mark said, \"Boo!\"\n""

Line=37, token=STRING, value=""Hi/n\n""
```

[Σχολιασμός]

Παρατηρούμε ότι ο λεκτικός αναλυτής λειτουργεί σωστά στα ακόλουθα παραδείγματα, καθώς οτιδήποτε δεν αναγνωρίζεται από την γλώσσα Uni-C θεωρείται unknown token.

```
// ====== Test case #3 INTEGERS ======
Line=40, token=INTEGER, value="0"

Line=41, token=INTEGER, value="3"

Line=42, token=INTEGER, value="214748"

Line=43, token=UNKNOWN TOKEN, value="-50"

Line=44, token=INTEGER, value="0x4F"

Line=45, token=INTEGER, value="0X88AA"

Line=46, token=INTEGER, value="063"

Line=47, token=INTEGER, value="00"
```

```
	Line=48, token=INTEGER, value="01"

	Line=49, token=UNKNOWN TOKEN, value="09"

	Line=50, token=UNKNOWN TOKEN, value="0XFGA9"

	Line=51, token=UNKNOWN TOKEN, value="01578"

	Line=52, token=UNKNOWN TOKEN, value="00xAFB1"

	Line=53, token=UNKNOWN TOKEN, value="-001"

	Line=54, token=UNKNOWN TOKEN, value="-0xAF01"

	Line=55, token=UNKNOWN TOKEN, value="0xff23"

	Line=56, token=UNKNOWN TOKEN, value="+01"

	Line=57, token=UNKNOWN TOKEN, value="+56"

	Line=58, token=UNKNOWN TOKEN, value="0x-FF"
```

[Σχολιασμός]

Στη γραμμή 71 παρατηρούμε ότι ο λεκτικός αναλυτής αναγνωρίζει λανθασμένα το 0 ως ακέραιο αριθμό. Με βάση τους κανόνες της Uni-C δεν θα έπρεπε να αναγνωρίζεται. Στη γραμμή 74 ο λεκτικός αναλυτής αναγνωρίζει τον ελληνικό χαρακτήρα Ε ως έγκυρο χαρακτήρα, αυτό δεν θα έπρεπε να συμβαίνει με βάση πάλι τους κανόνες της Uni-C.

```
// ====== Test case #4 FLOATS ======
	Line=61, token=FLOAT, value="9e8"

	Line=62, token=FLOAT, value="9E-8"
```

```
Line=63, token=FLOAT, value="3.14"

Line=64, token=UNKNOWN TOKEN, value="10.0"

Line=65, token=FLOAT, value="10.0001"

Line=66, token=FLOAT, value="5e-15"

Line=67, token=FLOAT, value="1e100"

Line=68, token=FLOAT, value="3.1E0"

Line=69, token=FLOAT, value="0e0"

Line=70, token=UNKNOWN TOKEN, value="-5.1e-100"

Line=71, token=INTEGER, value="0"

Line=72, token=UNKNOWN TOKEN, value="-0.5"

Line=73, token=UNKNOWN TOKEN, value="5.e1"

Line=74, token=FLOAT, value="1E1"

Line=75, token=UNKNOWN TOKEN, value="00.01"

Line=76, token=UNKNOWN TOKEN, value="0e01"

Line=77, token=UNKNOWN TOKEN, value="5e1.5e1"

Line=78, token=UNKNOWN TOKEN, value="1E1.2"
```

```
	Line=79, token=UNKNOWN TOKEN, value="6,20"


	Line=80, token=UNKNOWN TOKEN, value="0.0"
```

[Σχολιασμός]

Οποιοδήποτε σχόλιο είναι έγκυρο εμφανίζεται αλλά δεν επιστρέφεται ως token. Το "// hello" αναγνωρίζεται από τον λεκτικό αναλυτή επειδή οι ελληνικοί χαρακτήρες επιτρέπονται από το flex.

```
// ====== Test case #5 COMMENTS ======
// hello world!!
	Line=84, token=UNKNOWN TOKEN, value="/hello"
	Line=84, token=UNKNOWN TOKEN, value="world!!/"

	Line=85, token=UNKNOWN TOKEN, value="#"
	Line=85, token=IDENTIFIER, value="hello"
	Line=85, token=UNKNOWN TOKEN, value="world!"

// hello
	Line=87, token=IDENTIFIER, value="hello"
	Line=87, token=IDENTIFIER, value="world"

/* hello world!! */
//hello//world//!!
	Line=91, token=UNKNOWN TOKEN, value="\\"
	Line=91, token=IDENTIFIER, value="hello"
	Line=91, token=UNKNOWN TOKEN, value="world!!"

/*
 * hello
 * world
 * !!
 */
/*
```

```
hi

*/

       Line=96, token=UNKNOWN TOKEN, value="\*"

       Line=96, token=IDENTIFIER, value="hello"

       Line=96, token=UNKNOWN TOKEN, value="world!"

       Line=96, token=UNKNOWN TOKEN, value="*\"


/*//

//hi

//* hi */    Line=98, token=OPERATOR, value="/"


       Line=99, token=UNKNOWN TOKEN, value="%%"

       Line=99, token=IDENTIFIER, value="hi"


// hi */

/* hi //

// Αυτό είναι ένα σχόλιο

*/
```

[Σχολιασμός]

Τα keywords τα αναγνωρίζει ο λεκτικός αναλυτής ως identifier (λανθασμένα) μαζί με τον διαχωριστή “;”. Δεν αναγνωρίζει τον ειδικό χαρακτήρα : όπως βλέπουμε στα “case”. Οι τελεστές αναγνωρίζονται κανονικά.

```
// ====== Test case #6 KEYWORDS ======

       Line=105, token=IDENTIFIER, value="int"

       Line=105, token=IDENTIFIER, value="x"

       Line=105, token=OPERATOR, value="="

       Line=105, token=INTEGER, value="5"

       Line=105, token=DELIMITER, value=";"


       Line=106, token=IDENTIFIER, value="float"

       Line=106, token=IDENTIFIER, value="y"
```

```
Line=106, token=DELIMITER, value=";"


Line=107, token=IDENTIFIER, value="string"
Line=107, token=IDENTIFIER, value="str"
Line=107, token=OPERATOR, value="="
Line=107, token=STRING, value="""
Line=107, token=DELIMITER, value=";"

Line=108, token=IDENTIFIER, value="switch"
Line=108, token=IDENTIFIER, value="x"
Line=108, token=DELIMITER, value=";"

Line=109, token=IDENTIFIER, value="case"
Line=109, token=UNKNOWN TOKEN, value="1:"

Line=110, token=IDENTIFIER, value="break"
Line=110, token=DELIMITER, value=";"

Line=111, token=IDENTIFIER, value="case"
Line=111, token=UNKNOWN TOKEN, value="2:"

Line=112, token=IDENTIFIER, value="break"
Line=112, token=DELIMITER, value=";"

Line=113, token=UNKNOWN TOKEN, value="sizeof(str)"
Line=113, token=DELIMITER, value=";"

Line=114, token=IDENTIFIER, value="sizeof"
Line=114, token=DELIMITER, value=";"

Line=115, token=IDENTIFIER, value="double"
Line=115, token=OPERATOR, value="="
```

```
Line=115, token=FLOAT, value="3.14"
Line=115, token=DELIMITER, value=";"

Line=116, token=IDENTIFIER, value="const"
Line=116, token=IDENTIFIER, value="int"
Line=116, token=IDENTIFIER, value="z"
Line=116, token=OPERATOR, value="="
Line=116, token=INTEGER, value="5"
Line=116, token=DELIMITER, value=";"

Line=117, token=IDENTIFIER, value="struct"
Line=117, token=IDENTIFIER, value="Employee"
Line=117, token=DELIMITER, value=";"

Line=118, token=IDENTIFIER, value="if"
Line=118, token=IDENTIFIER, value="x"
Line=118, token=OPERATOR, value="=="
Line=118, token=INTEGER, value="0"

Line=119, token=IDENTIFIER, value="else"
Line=119, token=IDENTIFIER, value="x"
Line=119, token=OPERATOR, value="!="
Line=119, token=INTEGER, value="0"

Line=120, token=IDENTIFIER, value="return"
Line=120, token=INTEGER, value="0"
Line=120, token=DELIMITER, value=";"
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

[Σχολιασμός]

Στα ακόλουθα παραδείγματα ο λεκτικός αναλυτής λειτουργεί σωστά βάση των κανόνων ονοματοδοσίας στη γλώσσα Uni.

```
// ====== Test case #7 FINAL ======
	Line=123, token=IDENTIFIER, value="user"
	Line=123, token=DELIMITER, value=";"

	Line=124, token=UNKNOWN TOKEN, value="a=(b+c)"
	Line=124, token=DELIMITER, value=";"

	Line=125, token=STRING, value=""theo""

	Line=126, token=INTEGER, value="45"

	Line=127, token=FLOAT, value="4.8"

//hi
	Line=129, token=FLOAT, value="4.7"

	Line=130, token=STRING, value=""good moring""

	Line=131, token=FLOAT, value="4e2"

	Line=132, token=IDENTIFIER, value="good"
```

[Σχολιασμός]

Ο λεκτικός αναλυτής δεν αναγνωρίζει το break στη γραμμή 141 ως keyword.

```
// ====== Test case #8 FINAL ======
	Line=135, token=STRING, value=""pao""

/*20*/
	Line=138, token=FLOAT, value="8.5"
```

```
	Line=139, token=OPERATOR, value="+"

	Line=140, token=UNKNOWN TOKEN, value="+---"

	Line=141, token=IDENTIFIER, value="break"

	Line=142, token=IDENTIFIER, value="func"
```

[Σχολιασμός]

Τα αποτελέσματα του λεκτικού αναλυτή στα παρακάτω παραδείγματα είναι σωστά σύμφωνα με τα πρότυπα αναγνώρισης της Uni-C.

```
// ====== Test case #9 FINAL ======
	Line=145, token=UNKNOWN TOKEN, value="@#$"

	Line=146, token=UNKNOWN TOKEN, value="8,3"

	Line=147, token=UNKNOWN TOKEN, value="1_tzigger"

	Line=148, token=UNKNOWN TOKEN, value="/f"

	Line=149, token=UNKNOWN TOKEN, value="0.0"

	Line=150, token=FLOAT, value="5.001"
```

[Σχολιασμός]

Στη γραμμή 158 ο λεκτικός αναλυτής δεν αναγνωρίζει το keyword “while”.

```
// ====== Test case #10 OPERATORS ======
	Line=153, token=IDENTIFIER, value="x"
	Line=153, token=OPERATOR, value=">="
	Line=153, token=INTEGER, value="5"
```

```
Line=153, token=DELIMITER, value=";"


Line=154, token=INTEGER, value="5"
Line=154, token=OPERATOR, value="+"
Line=154, token=INTEGER, value="3"
Line=154, token=OPERATOR, value="="
Line=154, token=INTEGER, value="8"
Line=154, token=DELIMITER, value=";"

Line=155, token=UNKNOWN TOKEN, value="6/=3"
Line=155, token=DELIMITER, value=";"

Line=156, token=UNKNOWN TOKEN, value="x++"
Line=156, token=DELIMITER, value=";"

Line=157, token=UNKNOWN TOKEN, value="x*y"
Line=157, token=DELIMITER, value=";"

Line=158, token=IDENTIFIER, value="while"
Line=158, token=INTEGER, value="1"
Line=158, token=DELIMITER, value=";"

Line=159, token=IDENTIFIER, value="void"
Line=159, token=DELIMITER, value=";"

Line=160, token=IDENTIFIER, value="TRUE"
Line=160, token=OPERATOR, value="&&"
Line=160, token=IDENTIFIER, value="FALSE"
Line=160, token=OPERATOR, value="=="
Line=160, token=IDENTIFIER, value="FALSE"
Line=160, token=DELIMITER, value=";"
```

```
	Line=161, token=IDENTIFIER, value="TRUE"
	Line=161, token=OPERATOR, value="||"
	Line=161, token=IDENTIFIER, value="TRUE"
	Line=161, token=OPERATOR, value="=="
	Line=161, token=IDENTIFIER, value="TRUE"
	Line=161, token=DELIMITER, value=";"

	Line=162, token=UNKNOWN TOKEN, value="y--"
	Line=162, token=DELIMITER, value=";"

	Line=163, token=UNKNOWN TOKEN, value="&bit"
```

[Σχολιασμός]

Ο λεκτικός αναλυτής δεν αναγνωρίζει τα keywords “switch”, “case”, “break”, “return”.

```
// ====== Test case #11 CODE ======
	Line=166, token=IDENTIFIER, value="switch"
	Line=166, token=INTEGER, value="1"

	Line=167, token=IDENTIFIER, value="case"
	Line=167, token=INTEGER, value="1"

	Line=168, token=IDENTIFIER, value="break"
	Line=168, token=DELIMITER, value=";"

	Line=169, token=IDENTIFIER, value="case"
	Line=169, token=INTEGER, value="2"

	Line=170, token=IDENTIFIER, value="break"
	Line=170, token=DELIMITER, value=";"

	Line=171, token=IDENTIFIER, value="return"
```

```
      Line=171, token=INTEGER, value="0"
      Line=171, token=DELIMITER, value=";"
```

[Σχολιασμός]

Ο λεκτικός αναλυτής δεν αναγνωρίζει τα keywords.

```
// ====== Test case #12 CODE ======
      Line=174, token=IDENTIFIER, value="int"
      Line=174, token=IDENTIFIER, value="x"
      Line=174, token=OPERATOR, value="+"
      Line=174, token=IDENTIFIER, value="y"
      Line=174, token=OPERATOR, value="="
      Line=174, token=IDENTIFIER, value="z"
      Line=174, token=DELIMITER, value=";"

      Line=175, token=IDENTIFIER, value="var"
      Line=175, token=IDENTIFIER, value="x"
      Line=175, token=OPERATOR, value="+"
      Line=175, token=IDENTIFIER, value="y"
      Line=175, token=OPERATOR, value="="
      Line=175, token=IDENTIFIER, value="z"
      Line=175, token=DELIMITER, value=";"

      Line=176, token=IDENTIFIER, value="for"
      Line=176, token=IDENTIFIER, value="i"
      Line=176, token=OPERATOR, value="="
      Line=176, token=INTEGER, value="1"
      Line=176, token=DELIMITER, value=";"
      Line=176, token=IDENTIFIER, value="i"
      Line=176, token=OPERATOR, value="<="
      Line=176, token=IDENTIFIER, value="N"
      Line=176, token=DELIMITER, value=";"
      Line=176, token=UNKNOWN TOKEN, value="i++"
```

```
Line=176, token=IDENTIFIER, value="do"


Line=177, token=IDENTIFIER, value="if"
Line=177, token=IDENTIFIER, value="i"
Line=177, token=OPERATOR, value="%"
Line=177, token=INTEGER, value="2"
Line=177, token=OPERATOR, value="=="
Line=177, token=INTEGER, value="0"
Line=177, token=IDENTIFIER, value="then"

Line=178, token=IDENTIFIER, value="return"
Line=178, token=INTEGER, value="0"
Line=178, token=DELIMITER, value=";"

Line=179, token=IDENTIFIER, value="else"

Line=180, token=IDENTIFIER, value="return"
Line=180, token=INTEGER, value="1"
Line=180, token=DELIMITER, value=";"

Line=181, token=IDENTIFIER, value="end"
Line=181, token=IDENTIFIER, value="if"
Line=181, token=DELIMITER, value=";"

Line=182, token=IDENTIFIER, value="end"
Line=182, token=IDENTIFIER, value="for"
Line=182, token=DELIMITER, value=";"
```

[Σχολιασμός]

Στην γραμμή 185 τα λεξήματα “pi” (identifier), “=” (operator), “3.14” (float), εξαιτίας της απουσίας ενός διαχωριστή (κενό π.χ.), δεν αναγνωρίζονται ως έγκυρες λεκτικές μονάδες της Uni-C ξεχωριστά.

```
// ====== Test case #13 CODE ======

      Line=185, token=IDENTIFIER, value="const"

      Line=185, token=IDENTIFIER, value="double"

      Line=185, token=UNKNOWN TOKEN, value="pi=3.14"

      Line=185, token=DELIMITER, value=";"


      Line=186, token=IDENTIFIER, value="struct"

      Line=186, token=IDENTIFIER, value="Math"

      Line=186, token=IDENTIFIER, value="m1"

      Line=186, token=DELIMITER, value=";"


      Line=187, token=UNKNOWN TOKEN, value="m1.count"

      Line=187, token=OPERATOR, value="="

      Line=187, token=FLOAT, value="3.14"

      Line=187, token=OPERATOR, value="*"

      Line=187, token=INTEGER, value="2"

      Line=187, token=DELIMITER, value=";"


      Line=188, token=UNKNOWN TOKEN, value="m1.length"

      Line=188, token=OPERATOR, value="="

      Line=188, token=FLOAT, value="3.14"

      Line=188, token=OPERATOR, value="+"

      Line=188, token=INTEGER, value="2"

      Line=188, token=OPERATOR, value="/"

      Line=188, token=INTEGER, value="5"

      Line=188, token=DELIMITER, value=";"
```

[Σχολιασμός]

Το ίδιο σφάλμα με την αναγνώριση των keywords ως identifiers.

```
// ====== Test case #14 CODE/COMMENTS ======
       Line=191, token=IDENTIFIER, value="func"
       Line=191, token=IDENTIFIER, value="calculate_perimeter_of_table"
       Line=191, token=IDENTIFIER, value="void"


       Line=192, token=IDENTIFIER, value="int"
       Line=192, token=IDENTIFIER, value="length"
       Line=192, token=OPERATOR, value="="
       Line=192, token=INTEGER, value="5"
       Line=192, token=DELIMITER, value=";"
// Length of table
       Line=193, token=IDENTIFIER, value="int"
       Line=193, token=IDENTIFIER, value="width"
       Line=193, token=OPERATOR, value="="
       Line=193, token=INTEGER, value="10"
       Line=193, token=DELIMITER, value=";"
// Width of table
       Line=194, token=IDENTIFIER, value="int"
       Line=194, token=IDENTIFIER, value="x"
       Line=194, token=DELIMITER, value=";"



       Line=196, token=IDENTIFIER, value="x"
       Line=196, token=OPERATOR, value="="
       Line=196, token=IDENTIFIER, value="length"
       Line=196, token=OPERATOR, value="*"
       Line=196, token=INTEGER, value="2"
       Line=196, token=OPERATOR, value="+"
       Line=196, token=IDENTIFIER, value="width"
       Line=196, token=OPERATOR, value="*"
```

```
	Line=196, token=INTEGER, value="2"

	Line=196, token=DELIMITER, value=";"

// Perimeter of table

	Line=197, token=IDENTIFIER, value="return"

	Line=197, token=IDENTIFIER, value="x"

	Line=197, token=DELIMITER, value=";"


	Line=198, token=END_OF_FILE, value=""
```

5. Προβλήματα και τρόποι αντιμετώπισης

Πρόβλημα αντιμετωπίσαμε κυρίως στην σειρά προτεραιότητας των κανόνων στον κώδικα FLEX. Συγκεκριμένα, τον χαρακτήρα αλλαγή γραμμής (\n), ο λεκτικός αναλυτής το αναγνώριζε και επέστρεφε ως NEWLINE token και ως UNKNOWN TOKEN. Οι κανόνες είχαν την εξής σειρά και δομή.

```
NEWLINE		[\n]

UNKNOWN		.
```

Γνωρίζοντας ότι η έκφραση του UNKNOWN αναγνωρίζει οποιονδήποτε χαρακτήρα πέρα από την αλλαγή γραμμής, μας προκάλεσε έντονο προβληματισμό το γεγονός ότι επέστρεφε τον χαρακτήρα και ως UNKNOWN TOKEN.

Το πρόβλημα αντιμετωπίστηκε με την αντικατάσταση της κανονικής έκφρασης του κανόνα UNKNOWN με την παρακάτων

```
[^ \t\r\n;]+
```

Η έκφραση αναγνωρίζει οποιονδήποτε χαρακτήρα πέρα από τους χαρακτήρες διαφυγής tab, κενό, αλλαγή γραμμής, αλλαγή σειράς, και το ερωτηματικό. Παρόλο που τρέχουμε σε τερματικό Linux, χρησιμοποιήσαμε και το \r για την αλλαγή γραμμής.

Άλλο πρόβλημα που αντιμετωπίσαμε ήταν στην αναγνώριση των keywords όπως φαίνονται και στα σχόλια των περιπτώσεων ελέγχου. Τρόπος αντιμετώπισης δεν βρέθηκε λόγω περιορισμένου χρόνου για την υλοποίηση του μέρους αυτού. Τα keywords αναγνωρίζονται ως identifiers.

6. Ρητή αναφορά ελλείψεων

Η αναφορά περιλαμβάνει με ελλείψεις αυτά που απαιτούνται:

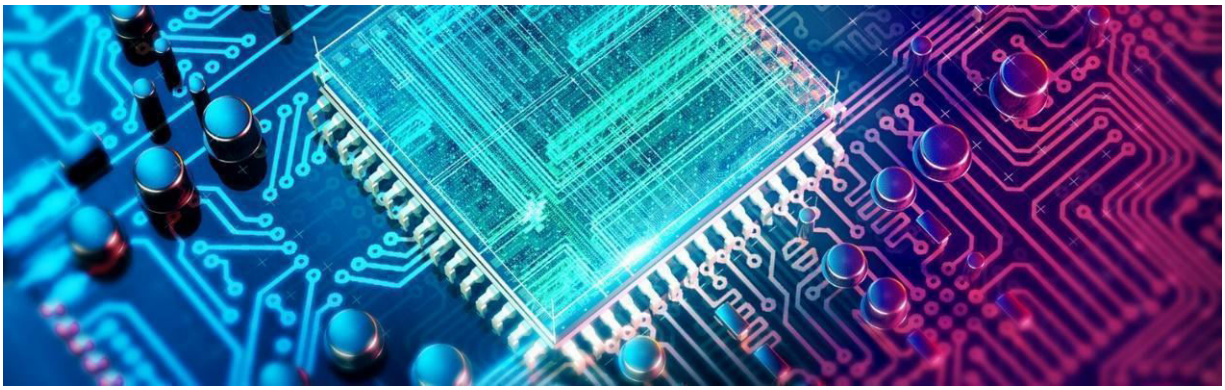
- Πίνακας με τα tokens
- Κώδικας FLEX
- Περιπτώσεις ελέγχου
- Σχολιασμός αποτελεσμάτων
- Προβλήματα που αντιμετωπίσαμε και τρόποι αντιμετώπισης

Ο κώδικας εκτελέστηκε σε περιβάλλον Windows με εγκατάσταση Windows Subsystem Linux (WSL). Το τερματικό που έτρεξε η ομάδα τον κώδικα ήταν σε Ubuntu 22.04 LTS, συνεπώς οι εντολές που χρησιμοποιήσαμε για την εκτέλεση του κώδικα είναι οι εξής:

```
make
```

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Σας ευχαριστούμε για την προσοχή σας.



ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ