Blue text on a black background

Description automatically generated

***Σχολή Μηχανικών***

***Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών***

***Εργαστήριο «Μεταγλωττιστές»***

***Μέρος Α-3 : Συμπλήρωση πρότυπου κώδικα FLEX***

***Ημερομηνία Αποστολής: 1/5/2024***

***Τμήμα Β2 - Ομάδα 2***

*ΚΟΝΤΟΥΛΗΣ ΔΗΜΗΤΡΙΟΣ 21390095*

*ΜΕΝΤΖΕΛΟΣ ΑΓΓΕΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ 21390132*

*ΒΑΡΣΟΥ ΕΥΦΡΟΣΥΝΗ 21390021*

*ΓΚΙΟΖΙ ΕΝΤΕΡΙΣΑ* *21390041*

*ΑΛΕΞΟΠΟΥΛΟΣ ΛΕΩΝΙΔΑΣ 21390006*

**Contents**

[**1. *Εισαγωγή* 3**](#_Toc165644729)

[**2. *Τεκμηρίωση* 3**](#_Toc165644730)

[**2.1 Ανάλυση είσοδου & εξόδου λεκτικού αναλυτή 3**](#_Toc165644731)

# 1. Εισαγωγή

Το παρών έγγραφο αναφέρεται στην ανάπτυξη και υλοποίηση του λεκτικού αναλυτή με το εργαλείο FLEX. Ο λεκτικός αναλυτής είναι ένα εργαλείο με το οποίο μπορούμε να εντοπίσουμε τις λεκτικές μονάδες της γλώσσας Uni-C. Οι λεκτικές μονάδες και οι ιδιότητες τους έχουν αναλυθεί ιδιαίτερα στο Μέρος Α-2 (Part A-2) της εργασίας, στο οποίο έγινε εκπόνηση των κανονικών εκφράσεων καθώς και των πεπερασμένων αυτόματων (σχέδια & κώδικας FSM) των λεκτικών μονάδων. Στο κεφάλαιο της τεκμηρίωσης, αναλύεται η λειτουργία του λεκτικού αναλυτή αλλά και τυχόν προβλήματα που προέκυψαν κατά την λεκτική ανάλυση, τα οποία μας έδωσαν λανθασμένα αποτελέματα και χρειάστηκαν περαιτέρω αναζήτηση. Επιπρόσθετα, θα γίνει ανάλυση του αρχείου με τις λεκτικές μονάδες το οποίο δόθηκε ως είσοδος στον λεκτικό αναλυτή, καθώς και το αρχείο εξόδου με τα αποτελέσματα που παράχθηκε από αυτόν.

# 2. Τεκμηρίωση

Παρακάτω ακολουθεί η ανάλυση τις λειτουργίας του λεκτικού αναλυτή. Για την περιγραφή της ορθής λειτουργίας του λεκτικού αναλυτή, θα γίνει αρχικά εξήγηση του αρχείου εισόδου το οποίο δόθηκε ως είσοδο σε αυτόν και έπειτα θα γίνει περιγραφή της εξόδου καθώς και σχολιασμός των αποτελεσμάτων. Το πρόγραμμα που αναπτύχθηκε ως είσοδος για τον λεκτικό αναλυτή είναι C – like πρόγραμμα, δηλαδή οι εντολές αντιπροσωπεύουν αυτές τις γλώσσας C. Είναι σημαντικό να αναφερθεί πως για την υλοποίηση του λεκτικού αναλυτή είναι απαραίτητοτ το αρχείο token.h (header file), το οποίο περιέχει την αντίστοιχη τιμή επιστροφής για κάθε token. Τα tokens ουσιαστικά είναι ονόματα τα οποία αντιπροσωπεύουν τις κανονικές εκφράσεις που έχουν τοποθετηθεί ως προς αναγνώριση (εξηγείται και εντός του αρχείου του λεκτικού αναλυτή).

## 2.1 Ανάλυση είσοδου & εξόδου λεκτικού αναλυτή

int main(int argc, char\*\* argv) {

// testing numbers

int num = 5; // whitespaces are ignored

int num1 = -10; // τhis is not recognized as a number, ιt's an operator and a number

int num2 = 0x14FAC2;

int num3 = 0;

int num4 = 012442562378; // the analyzer will separate the 8 from the other part of the number

float result = 3.14e-10;

float b = 1.1e3.14 // this is not a valid number

// Arithmetical operators

result += num1 + (num2 \* num3);

result -= num2 - (num3 / num1);

#modulo /= num1 % num1;

// Comparison operators and string literals

if (num1 >= num2) {

printf("num1 is greater than num2\n");

} else if (num1 <= num2) {

printf("num1 is less than num2\\");

} else if (num1 !== num2) {

printf("numbers are not equal\"");

}

// testing braces

char str1[24] = "Compilers Lab";

else if (num1 < 0 | num2 != 0 || num3 == 0)

{

do {

--result;

result++;

printf("inside loop /n"); // /n is not a valid operator, \n is

if(result == 0) break;

} while(!num4 == 0 || num3 +- = 0);

}

// testing braces

char str1[] = "Compilers Lab";

/\* the combination as a whole won't be recognized as an identifier.

~ is an unknown token and @ is a symbol. \*/

~var@ = 5;

/\* This is an unterminated multi-line comment

that will cause the analyzer to break because of EOF.

Because of this, in the output of the lexical analyzer (output.txt)

we will see that the brackets of the main function never close.

return 0;

}

Επειδή το αρχείο εισόδου είναι σημαντικού μεγέθους, θα γίνει ανάλυση του σε όσο το δυνατόν καλύτερα χωρισμένα κομμάτια γίνεται. Το αρχείο αυτό βρίσκεται διαθέσιμο και στο τελικό παραδοτέο .zip με όνομα input.txt. Ξεκινήσαμε γράφοντας μία συνάρτηση main, έτσι ώστε να τεστάρουμε για αρχή τον εντοπισμό των identifiers, των keywords, του συμβόλου “,” αλλά και τις παρενθεσεις. Έπειτα ξεκινλησαμε από τη δοκιμή κάθε είδους αριθμού (δεκαδικοί, δεκαεξαδικοί, ακέραιοι, οκταδικοί, εκθετικοί).

Ξεκινόντας από τους ακεραίους δοκιμάσαμε ένα απλό ακέραιο, στον οποίο όμως τοποθετήσαμε και μερικά έξτρα spaces για να ελέγξουμε και τον εντοπισμό των whitespace από τον αναλυτή. Αυτό που περιμένουμε στην έξοδο είναι η αγνόηση των whitespaces από τον αναλυτή και η αναγνώριση του αριθμού. Συνεχίζοντας με τους αριθμούς δοκιμάσαμε να βάλουμε έναν αρνητικό αριθμό. Στην ουσία όμως οι αρνητικοί αριθμοί δεν αναγνωριζόνται ως λεκτικές μονάδές από μόνοι τους, αναγνωρίζονται ως τον συνδυασμό ενός operator συνοδευόμενο από έναν αριθμό. Οπότε περιμένουμε να το δούμε αυτό στην έξοδο μας μετέπειτα. Περαιτέρω δοκιμάσαμε έναν δεκαεξαδικό αριθμό (num2) καθώς και το 0 μόνο του (num3). Στην Πέμπτη περίπτωση (num4), εισάγουμε έναν οκταδικό αριθμό, ο οποίος όμως στο τελευταίο του ψηφίο έχει το 8, το οποίο δεν είναι επιτρεπτό στους οκταδικούς. Οπότε στην έξοδο μας περιμένουμε να δούμε χωρισμένο τον οκταδικό αριθμό από το 8. Και οι δύο όμως αναγνωρίζονται ως ακέραιοι (integers) . Το 3.14e-10 θα ανγνωριστεί κανονικά ως float αριθμός. Το 1.1e3.14 δεν μπορεί να αναγνωριστεί ολόκληρο, επειδή το 1.1e3 είναι όντως αριθμός, αλλά το .14 δεν είναι. Συγκεκριμένα το σύμβολο . (τελεία) επιστρέφεται ως unknown token, και ο αριθμός ως απλά ένα integer.

Μετέπειτα δοκιμάσαμε τους αριθμητικούς τελεστές. Αρχικά βλέπουμε τον τελεστή += ο οποίος αποτελεί έναν τελεστή μόνος του και όχι ως δύο διαφορετικούς τελεστές + και =. Σε αυτή καθώς και στην επόμενη σειρά στον κώδικα, έχουμε τοποθετήσει και παρενθέσεις, οπότε περιμένουμε να δούμε και αυτές στην έξοδο. Στο τρίτο παράδειγμα (#modulo), ο αναλυτής περιμένουμε να μην το αναγνωρίσει ως ένα identifier, αλλά να ξεχωρίσει το σύμβολο # από το identifier modulo. Παρακάτω ελέγχουμε τους συγκριτικούς τελεστές τους οποίους έχουμε τοποθετήσει σε δομές ελέγχου if, εντός των οποίων τεστάρουμε κάποια string literals σε συνδυασμό με Escape Sequence characters. Οπότε σε αυτό το κομμάτι κώδικα, περιμένουμε να δούμε τους τελεστές αναγνωρισμένους από τον αναλυτή, εκτός από τον τελευταίο (!==), ο οποίος πρέπει να αναγνωρισθεί ξεχωριστά ως ‘!=’ και ‘=’ ξεχωριστά.

Στην τελευταία γραμμή τοποθετήσαμε ένα identifier ακολουθούμε από braces τα οποία περιέχουνε αριθμό μέσα ( [24] ), δηλαδή σε μορφή πίνακα, έτσι ώστε να ελέγξουμε ότι ο λεκτικός αναλυτής αναγνωρίζει ξεχωριστά τα braces και τον αριθμό.

Παρακάτω βρίσκεται το υπόλοιπο κομμάτι του αρχείου εισόδου στο οποίο έχουμε τοποθετήσει λοιπές λεκτικές μονάδες.

// Logical operators and arithmetic operators

if (num1 > 0 &&& num2 > 0) {

printf("Both num1 and num2 are positive\n");

}

else if (num1 < 0 | num2 != 0 || num3 == 0)

{

do {

--result;

result++;

printf("inside loop /n"); // /n is not a valid operator, \n is

if(result == 0) break;

} while(!num4 == 0 || num3 +- = 0);

}

/\* the combination as a whole won't be recognized as an identifier.

~ is an unknown token and @ is a symbol. \*/

~var@ = 5;

/\* This is an unterminated multi-line comment

that will cause the analyzer to break because of EOF.

Because of this, in the output of the lexical analyzer (output.txt)

we will see that the brackets of the main function never close.

return 0;

}

Συνεχίζουμε λοιπόν με τον έλεγχο των λογικών τελεστών. Στην αρχική δομή if, ο συνδυασμός &&& δεν αποτελεί κάποιο είδος τελεστή, οπότε πρέπει να αναγνωριστεί ξεχωριστά ως ‘&&’ και ‘&’. Αντίστοιχα στην δομή else if βλέπουμε το σύμβολο ‘|’ το οποίο πρέπει δεν αποτελεί κάποιο σύμβολο της αλφαβήτου της γλώσσας (unknown token), ενώ ο δεύτερος είναι οντως σωστός και στην ουσία παριστά μία λογική πράξη OR.