ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

Σχολή Μηχανικών

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Εργαστήριο «Μεταγλωττιστές»

Μέρος Β-2: Σύνδεση κώδικα flex με κώδικα bison

Μέρος Β-3: Διαχείριση λεκτικών και συντακτικών προειδοποιητικών λαθών

Ημερομηνία Αποστολής: 17/5/2024

Τμήμα Β2 - Ομάδα 2

ΚΟΝΤΟΥΛΗΣ ΔΗΜΗΤΡΙΟΣ 21390095

ΜΕΝΤΖΕΛΟΣ ΑΓΓΕΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ 21390132

ΒΑΡΣΟΥ ΕΥΦΡΟΣΥΝΗ 21390021

ΑΛΕΞΟΠΟΥΛΟΣ ΛΕΩΝΙΔΑΣ 21390006

Περιεχόμενα

1. Εισαγωγή

Το παρών έγγραφο αποτελεί η κωδικοποίηση και σχεδιασμού ενός Συντακτικού αναλυτή με την γεννήτρια Bison. Ειδικότερα, επεκτείναμε την λογική που υλοποιήσαμε για το Μέρος Α-3 της λεκτικής ανάλυσης με την γεννήτρια Flex (αναγνώριση λεκτικών μονάδων) για να μπορούμε πλέον να αναγνωρίζουμε εκφράσεις με τον συνδυασμό των λεκτικών μονάδων. Δηλαδή, ο κώδικας Bison που υλοποιήσαμε διαπιστώνει την σωστή γραμματική της γλώσσας με είσοδο ένα πηγαίο κώδικα. Επιπλέον, ο Bison διαπιστώνει και την σημασιολογική ανάλυση δηλαδή εκτός αν γραμματικά είναι σωστή η έκφραση αλλά και επιπλέον αν μπορεί να οριστεί π.χ. απροσδιόριστες μεταβλητές, απροσδιόριστοι τύποι. Τέλος, ο κώδικας Bison κάνει και διαχείριση λεκτικών και συντακτικών προειδοποιητικών λαθών που ενημερώνει τον χρήστη που έγινε το λάθος.

Η δομή του ακόλουθου εγγράφου αποτελείται από τα κεφάλαια τεκμηρίωση της γραμματικής που υλοποιήσαμε στον κώδικα Bison, της παρουσίασης εξαντλητικών ελέγχων με διαφορετικά αρχεία εισόδου-εξόδου, της ανάλυσης προβλημάτων ελλείψεων που αντιμετωπίσαμε και υπάρχουν στον κώδικα και του επίλογου που κλείνουμε με κάποια τελικά γενικά σχόλια σχετικά με την υλοποίηση της εργασίας.

2. Τεκμηρίωση

2.1 Γραμματική (Μέρος Β-2)

Σε αυτό το κεφάλαιο θα αναλύσουμε γενικά όλα τα στάδια της γραμματικής από την λεκτική ανάλυση που δημιουργήσαμε ενδεικτικά στο Α-3, στην συντακτική ανάλυση και τέλος στην σημασιολογική ανάλυση. Η λεκτική ανάλυση γίνεται στο flex κώδικα ενώ η συντακτική με την σημασιολογική στον κώδικα bison.

2.1.1 Λεκτική Ανάλυση

Στην λεκτική ανάλυση, δεν έχουν αλλάξει πολλά από ότι εξηγήσαμε στο Α-3. Η σημαντική αλλαγή για να συνδέεται με τον κώδικα του bison ήταν στο κομμάτι των ορισμών για τον κώδικα να αλλάξουμε το header από το δικό μας ορισμένο «token.h» στο header που δημιουργεί ο bison (syntax_analyzer.tab.h). Επίσης έχουμε δηλώσει τις μεταβλητές lex_warn, cor_words, inc_words οι οποίοι χρησιμεύουν στην καταμέτρηση των λέξεων και στα προειδοποιητικά σφάλματα (που θα αναλυθούν στην συνέχεια). Στην συνέχεια, δίνονται οι κανονικές εκφράσεις για το κάθε token (που έχουν οριστεί και από τον bison με %token και ο οποίος έχει αναθέσει κωδικό στο header file) και η γραμμή «%x REALLYEND error» δηλώνει δύο αποκλειστικές συνθήκες εκκίνησης REALLYEND και error. Με αυτήν την γραμμή όταν βρεθεί σε μία από αυτές τις δύο καταστάσεις αλλάζει τη συμπεριφορά του για να χειρίζεται είτε το σενάριο τέλους αρχείου EOF (REALLYEND) ή σφάλματα (error) και διαχείριση αυτών. Για κάθε token που έχουμε ορίσει και κανονική έκφραση αυξάνουμε τον μετρητή για τις σωστές λέξεις και επιστρέφουμε το αντίστοιχο token (για να τα αναγνωρίζει ο bison). Και γενικά τα υπόλοιπα θα αναλυθούν περισσότερο στο κεφάλαιο 2.2 του έγγραφου.

2.1.2 Συντακτική Ανάλυση

Γενικά, η συντακτική ανάλυση έχει αναλυθεί σε πολλές γραμματικές που αφορούν την δήλωση μεταβλητών, τις Built-in συναρτήσεις της Uni-C, τις δηλώσεις συναρτήσεων χρήστη, την δήλωση απλών εκφράσεων και τις σύνθετες δηλώσεις (if, while, for). Αυτές τις γραμματικές τις έχουμε ενσωματώσει στην γενική γραμματική valid που είναι η γραμματική που τελειώνει το πρόγραμμα (program valid). Είναι δηλαδή οι γραμματικές declaration (δήλωση μεταβλητών), func_call (bulit_in συναρτήσεις και κλήση συνάρτησης χρήστη), assignment (Αναθέσεις τιμών σε αναγνωριστικά), func_decl (δήλωση συνάρτηση χρήστη), if_while_grammar και for_grammar (σύνθετες δηλώσεις). Επιπλέον, η valid έχει και την λεκτική μονάδα EOP που συμβολίζει το τέλος του αρχείου, και οπότε καλεί την συνάρτηση print_report() για το τύπωμα τον σωστών ή λανθασμένων λεκτικών και συντακτικών λέξεων στο τέλος της συντακτικής ανάλυσης.

Στις γραμματικές keyword, operator, num, var και str ουσιαστικά δεχόμαστε τα tokens για το κάθε ένα αντίστοιχα και αντιγράφουμε το κείμενο που δόθηκε και το αποδίδουμε στη σημασιολογική τιμή του κανόνα ($$). Το κάνουμε αυτό γιατί π.χ. για τους operators (και τα keywords αντίστοιχα) στον γραμματικό κανόνα operator_val κάνουμε την αντιστοίχιση σε κωδικούς (id) τον κάθε operator (για την σημασιολογική ανάλυση που χρειάζεται στους έπειτα κανόνες).

Σε γενικές γραμμές στον κώδικα, έχουμε και βοηθητικούς κανόνες (όπως π.χ help_int, help_float κτλ) που χρησιμεύουν στην παράθεση ίδιων τύπων λέξεων να παρατίθενται είτε μόνες τους είτε με κόμματα (SYMBOL) όταν

είναι δύο και άνω. Αυτούς τους τύπους γραμματικών κανόνων τους θέλουμε είτε στους πίνακες είτε στην ομαδική ανάθεση τιμών κτλ. Οπότε γενικά χρησιμοποιούνται σε πολλές γραμματικές της Uni-C.

Η γραμματική expr ορίζει τις πράξεις και τις συγκρίσεις μεταξύ αριθμών, η οποία στην συνέχεια θεωρείται και γραμματική num (που περιέχει όλους τους αριθμούς). Αυτό το κάνουμε για να δέχεται π.χ. η δομή if και while και αριθμούς (που στην C αν είναι μεγαλύτερες του 1 σημαίνουν true) και πράξεις και συγκρίσεις.

Για τις builit-in συναρτήσεις (scan, len, print) συνδυάζουμε τους γραμματικούς κανόνες που τις αναγνωρίζουν γιατί έχουν κοινές γραμματικές (π.χ η print για το ένα όρισμα της μοιάζει με την len και την scan). Ανάλογα με τον κωδικό που δίνεται δηλαδή για το keyword ξεχωρίζουμε ποια είναι ποια και χωρίζουμε και συνδυάζουμε κατάλληλα τις γραμμικές στις scan_len_print, cmp_print και print οι οποίες όλες μαζί ενσωματώνονται στην γραμματική func_call. Στην func_call επίσης, έχουμε και τις διάφορες γραμματικές για τις κλήσεις συναρτήσεων που είναι δηλωμένες για τον χρήστη.

Τέλος, για τον ίδιο λόγο με πριν έχουμε συνδυάσει την γραμματική για την δομή if με την δομή while (if_while_grammar) γιατί ορίζονται με τον ίδιο τρόπο. Μόνο η if έχει τον παραπάνω γραμματικό κανονικά για την else την οποία την ελέγχουμε σημασιολογικά.

Ο κάθε γραμματικός κανόνας αναλύεται και μέσα σε σχόλια στον κώδικα αλλά γενικά για τον σχεδιασμό και για την λειτουργεία των υπόλοιπων βασίστηκε στους πάνω πολύ βασικούς κανόνες της γραμματικής μας που τις χρησιμοποιούμε σε πολλά σημεία στο πρόγραμμά μας.

2.1.3 Σημασιολογική Ανάλυση

Η σημασιολογική ανάλυση αφορά την αναγνώριση της ορθής δομής του κώδικα. Ενώ η συντακτική ανάλυση εστιάζει στο πως συνδέονται μεταξύ τους τα διάφορα tokens, η σημασιολογική ανάλυση εξετάζει την σχέση μεταξύ αυτών. Ένα παράδειγμα το οποίο αφορά μία τέτοιου είδους ανάλυση, είναι η ορθότητα μεταξύ των τύπων. Δηλαδή αν μία μεταβλητή έχει δηλωθεί ως ακέραιος (int), τότε δεν είναι εφικτό να της ανατεθεί τιμή η οποία είναι πραγματικός αριθμός (float). Ένα άλλο παράδειγμα αποτελούν οι σημασιολογικές διαφορές μεταξύ των ενσωματωμένων συναρτήσεων print και len. Τέτοιου είδους παραδείγματα θα αναλυθούν και παρακάτω, καθώς και ο τρόπος με τον οποίο διαβεβαιώνουμε ότι η ανάλυση αυτών πραγματοποιείται ορθά.

Οι γραμματικοί κανόνες που βρίσκονται στο σύμβολο declaration, αφορούν την δήλωση των μεταβλητών. Εδώ πραγματοποιούνται οι απαραίτητοι έλεγχοι, προκειμένου δηλώσεις όπως int a = 4.7 να είναι μη εφικτές. Αυτό υλοποιείται εντός των brackets του εκάστοτε γραμματικού κανόνα (κώδικας C), ελέγχοντας σε κάθε περίπτωση, ότι κάθε σύμβολο περιέχει την αντίστοιχη σωστή τιμή. Πραγματοποιούνται δηλαδή λογικοί έλεγχοι, έτσι ώστε ο αναλυτής να μην επιτρέπει την ύπαρξη του keyword int με την ανάθεση πραγματικής τιμής σε αυτό. Ομοίως λειτουργούν και όλοι οι υπόλοιποι γραμματικοί κανόνες, απλώς άλλοι γραμματικοί κανόνες αφορούν άλλες δομές και άλλα κομμάτια του κώδικα. Το ίδιο γίνεται και στους πίνακες. Αν δηλωθεί ένα πίνακας με όνομα arr του οποίου ο τύπος είναι ακέραιος, μία δήλωση του τύπου: int arr = [0.5, 1] δεν είναι εφικτή, καθώς θα πρέπει να υπάρχουν αποκλειστικά ακέραιοι εντός του πίνακα.

Ας δούμε τώρα και τις περιπτώσεις των built-in συναρτήσεων. Συγκεκριμένα, οι συναρτήσεις scan και len μπορούν να δεχθούν πίνακες (arrays) ως ορίσματα. Σημασιολογικά όμως, οι συναρτήσεις στην ουσία παράγουν διαφορετικά αποτελέσματα, οπότε αξιοποιούν τους πίνακες με διαφορετικό τρόπο. Στην πραγματικότητα, η print τυπώνει δεδομένα, ενώ η len υπολογίζει το μέγεθος ενός πίνακα ή μίας συμβολοσειράς. Στο τμήμα scan_len_print του προγράμματος μας, πραγματοποιούνται οι απαραίτητοι σημασιολογικοί έλεγχοι των δύο αυτών συναρτήσεων. Ενδεικτικά, εντός της print μπορεί να δοθεί το όνομα ενός πίνακα συνοδευμένο από ένα index, με το οποίο μπορούμε να εκτυπώσουμε την τιμή που βρίσκεται στη συγκεκριμένη θέση του πίνακα (π.χ arr[0]). Αυτό με τη χρήση της len δεν είναι εφικτό, καθώς η σύνταξη της πρέπει να έχει την εξής δομή: len(arr) ή len([1, 2, 3]).

Συνοψίζοντας, η σημασιολογική ανάλυση είναι μία ιδιαίτερα σημαντική διαδικασία η οποία πραγματοποιείται κατά τη μεταγλώττιση, η οποία αφορά την ορθότητα της σύνταξης όπως αναφέρθηκε και παραπάνω. Με βάση τα προαναφερθέντα, η σημασιολογική ανάλυση μας επιτρέπει να διεξάγουμε ελέγχους σχετικά με τα είδη μεταβλητών, την ορθότητα δομών ελέγχου και επανάληψης, αλλά και να δεχόμαστε ανατροφοδότηση σχετικά με τα διάφορα λάθη που περιέχει ο εκάστοτε κώδικας που δίνεται ως input.

2.2 Διαχείριση Προειδοποιητικών λαθών (Μέρος Β-3)

Στην συνέχεια, επεκτείναμε τον κώδικα Bison και Flex για να διαχειρίζεται σφάλματα κατά την εκτέλεση και να βγάζει τα αντίστοιχα errors (που έχουν οριστεί στον κώδικα flex) και warnings (που έχουν γίνει σε κάποιες γραμματικές). Ειδικότερα, έχουν υλοποιηθεί 6 διαφορετικού τύπου warnings. Για να γίνει αυτό χρειάστηκαμε έναν μετρητή στον συντακτικό αναλυτή για την αναγνώριση των warnings (par_warn).

Αρχικά στην γραμματική arr για να γίνουν δυο πίνακες concat ελέγχουμε αν το operator είναι – ($2 ==2 από την λίστα με τα operators). Αν είναι – τότε να πετάει μήνυμα “το concat απέτυχε και κρατάμε τον πρώτο πίνακα”. Αμέσως μετά αυξάνουμε τον μετρητή με τα warnings και τον μετρητή για τις λάθος εκφράσεις.

Στην γραμματική declaration έχει μπει μια καινούργια γραμματική η οποία μετά το declaration πχ int a = b έχει μπει ένας τελεστής δηλαδή int a = b ! ;. Αν γίνει αυτό τότε πετάει μήνυμα “ότι βρέθηκε operator στο declaration”. Αμέσως μετά αυξάνουμε τον μετρητή με τα warnings και τον μετρητή για τις λάθος εκφράσεις.

Στην γραμματική assignment που γίνεται η ισότητα μεταβλητής με πίνακα πχ x = [1,2];, αν ο χρήστης βάλει παραπάνω πίνακες από όσες μεταβλητές έχει τότε πετάει μήνυμα “ότι οι υπόλοιποι πίνακες δεν αντιστοιχήθηκαν στις μεταβλητές”. Αμέσως μετά αυξάνουμε τον μετρητή με τα warnings και τον μετρητή για τις λάθος εκφράσεις.

Στην γραμματική print έχουν μπει δυο προειδοποιητικά λάθη. Η print εκτυπώνει το περιεχόμενο που θα μπει στις παρενθέσεις όπως print(len(a));, αν ο χρήστης βάλει για το πρώτο λάθος βάλει τα keyword len cmp (κωδικοί 4 – 5) π.χ. print(len a) και μετά βάλει κατευθείαν την μεταβλητή να πετάξει μήνυμα “ότι δεν χρησιμοποιείτε σωστά η len/cmp” ενώ αν βάλει όλα τα υπόλοιπα keywords θα εκτυπώσει μήνυμα “ότι υπάρχει λάθος keyword μέσα στην print”. Και στα δύο προειδοποιητικά λάθη αυξάνουμε τον μετρητή με τα warnings και τον μετρητή για τις λάθος εκφράσεις.

Τέλος στην γραμματική func_decl έχει μπει μια καινούργια γραμματική στην οποία αν στην δήλωση συνάρτησης αμέσως μετά την λέξη func υπάρχει αριθμός και μετά το όνομα της συνάρτησης και τα ορίσματά της π.χ. func 5 calc (int a); να εκτυπώνει μήνυμα “Βρέθηκε αριθμός στην δήλωση συνάρτησης. Αμέσως μετά αυξάνω τον μετρητή με τα warnings και τον μετρητή για τις λάθος εκφράσεις.

3. Παρουσίαση Εξαντλητικών ελέγχων

3.1 Μέρος: Β-2

3.1.1 Δήλωση μεταβλητών (Declaration)

```
/* Correct Declarations */
int a;
double var;
float c;
int a = 5;
float b = 7.62;
double c = d;

int sum = a + b;
double average = sum/count;
float f = c/d;
const int id;
const int p = 2;
long float speed = 5.5;

int arr = [2, 4, 6, 8];
float f_array = [1.7, 4.7, 8.2];
long double price = price + price;

/* Incorrect Declarations */
int ;
float int a;

int a = 1.1;

int int_arr = [1, 5.4, 3];
float f_arr = [4, 3, 2.5];
int arr = [1, "var", 5];

int 1var;
float const v;
double 5.5 = var;
int long sum;
```

```
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
Valid Declaration
        Line 23 at lexeme ';' : syntax error
        Line 24 at lexeme ';' : Invalid declaration type
        Line 26 at lexeme ';' : Invalid declaration type
        Line 28 at lexeme '5.4' : syntax error
        Line 29 at lexeme '2.5' : syntax error
        Line 30 at lexeme '"var"' : syntax error
        Line 32 at lexeme ';' : syntax error
        Line 33 at lexeme ';' : Invalid declaration type
        Line 35 at lexeme 'var' : syntax error
        Line 37 at lexeme ';' : Invalid declaration type
#END-OF-FILE#
-------Report:-------
Correct Words: 154
Correct Expressions: 15
Incorrect Words: 0
Incorrect Expressions: 10
```

Παραπάνω παρατηρούνται το κομμάτι κώδικα που δόθηκε ως input (αριστερό μέρος), καθώς και το output που παράχθηκε από τον μεταγλωττιστή. Αρχικά, βλέπουμε διάφορες παραλλαγές δήλωσης μεταβλητών, οι οποίες είναι ορθές και επιτρέπονται στη Uni-C. Περαιτέρω, στο ίδιο πλαίσιο υπάρχουν και κάποιες μη επιτρεπτές δηλώσεις μεταβλητών. Στο output αντίστοιχα, βρίσκεται η άναλυση κάθε γραμμής. Εντοπίζοντας σωστή δήλωση, τυπώνει τη φράση Valid Declaration, αντιθέτως, αν εντοπίσει κάποιο λάθος, τυπώνει το είδος του λάθους, καθώς και σε ποιά γραμμή εντοπίσθηκε.

3.1.2 Κλησεις συναρτήσεων (built-in και δηλωμένες από τον χρήστη)

```
/* Correct Built in functions */
scan(x);
scan(var);

len([10, 20, 30, 40, 50]);
len("This is a test string");
len(arr);

cmp("test", "strings");
cmp(string1, string2);

print("Hello World");
print(myVar, "-", 50);

print(cmp(str1, str2));
print(len("Test string"));
print(len([10, 20]));
print(pin[0]);

/* Correct User defined function calls */
myfunc(10, 20, 30);
myfunc(1, a, "a");
myfunc();

/* Incorrect Built in functions */
scan(5);
scan(if);

len();
len(47);
len(str1, str2);
len(pin[0]);
cmp(1);
cmp(str1);
cmp(str1, str2, str3);
cmp(5, "a");

print(5);
print(len());
print(len(5));
print(len(10, 20));

print(cmp());
print(cmp, len);
print([1, 2, 3]);

/* Incorrect User defined function calls */
myfunc([1,2]);
```

```
Valid function call
Valid function call
Valid Function call
Valid function call
Valid function call
Valid function call
Valid function call
Valid function call
Valid function call
Valid function call
Valid function call
Valid function call
Valid function call
Valid Function call
Valid function call
Valid Function call
Valid function call
Valid function call
Valid function call
        Line 26 at lexeme ';' : syntax error
        Line 27 at lexeme ')' : syntax error
        Line 29 at lexeme ')' : syntax error
        Line 30 at lexeme ';' : syntax error
        Line 31 at lexeme ')' : Invalid function call
        Line 32 at lexeme ')' : Invalid function call

        Line 33 at lexeme ';' : syntax error
        Line 34 at lexeme ')' : Invalid function call
        Line 35 at lexeme ')' : Invalid function call
        Line 36 at lexeme ')' : Invalid function call
        Line 38 at lexeme ';' : syntax error
        Line 39 at lexeme ')' : syntax error
        Line 40 at lexeme '5' : syntax error
        Line 41 at lexeme '10' : syntax error
        Line 43 at lexeme ')' : syntax error
        Line 44 at lexeme ',' : syntax error
        Line 45 at lexeme ')' : Invalid function call

        Line 48 at lexeme '[' : syntax error
#END-OF-FILE#
-------Report:-------
Correct Words: 247
Correct Expressions: 19
Incorrect Words: 0
Incorrect Expressions: 18
```

Το εξής παράδειγμα αφορά τις κλήσεις συναρτήσεων, τόσο για τις built-in όσο και για τις user-defined. Πραγματοποιούνται απλοί έλεγχοι, όπως scan(x) ή len(arr), αλλά και πιό σύνθετοι όπως στη περίπτωση της κλήσης print(cmp(str1, str2)) όπου βρίσκονται συναρτήσεις η μία εντός της άλλης. Παρατηρούμε ότι οι κλήσεις των συναρτήσεων που είναι συντακτικά ορθές εντοπίζονται ορθά από τον μεταγλωττιστή, όπως φαίνεται και στο output. Προχορώντας, δοκιμάζονται οι κλήσεις συναρτήσεων που δεν είναι συντακτικά ορθές, με βάση τις προδιαγραφές της γλώσσας. Κλήσεις όπως, scan(5), len(str1, str2) και print([1, 2, 3]) εντοπίζονται ως λανθασμένες από τον μεταγλωττιστή. Αυτό παρατηρείται και στο τελικό output.

3.1.3 Αναθέσεις μεταβλητών (Assignment)

```
/* Correct Assignments */
x1 = 0;
x1, x2 = 0, 1;
x1, arr = 0, [1, 2, 3];
x1, arr, string = 0, [1, 2, 3], "HELLO";

f1, f2 = 0.1, 0.2;
f1, arr2 = 0.5, [0.2, 0.4, 0.6];
f1, arr2, str2 = 0.5, [0.3, 0.5, 0.7],
"HELLO";

/* Incorrect Assignments*/
x1, x2 = 0;
f1, arr = 0.1, ["test", 5];

x = 0, 0.5, "HELLO";

list_of_nums = [1, 2, , 4];
comp = a < > b;
```

```
Valid assignment
Valid assignment
Valid assignment
Valid assignment
Valid assignment
Valid assignment
Valid assignment
        Line 12 at lexeme ';' : Invalid assignment
        Line 13 at lexeme '5' : syntax error
        Line 15 at lexeme '"HELLO"' : syntax error
        Line 17 at lexeme ',' : syntax error
        Line 18 at lexeme 'b' : syntax error
#END-OF-FILE#
-------Report:-------
Correct Words: 128
Correct Expressions: 7
Incorrect Words: 0
Incorrect Expressions: 5
```

Στα παραπάνω πλαίσια παρατηρείται η ανάλυση τόσο των απλών, όσο και των σύνθετων αναθέσεων μεταβλητών. Στις λανθασμένες δηλώσεις, ελέγχουμε περιπτώσεις όπως ανάθεση δύο μεταβλητών στην ίδια τιμή, ανάθεση πίνακα που περιέχει τιμές διαφορετικού τύπου, καθώς και ανάθεση πολλών, ξεχωριστών τιμών σε μία μόνο μεταβλητή. Βλέπουμε για παράδειγμα ότι στη γραμμή 13, ο μεταγλωττιστής εντοπίζει ορθά το λάθος στην εντολή f1, arr = 0.1, [“test”, 5], το οποίο είναι το 5 και αποτελεί λάθος τύπο.

3.1.4 Παραδείγματα ορισμένων συναρτήσεων (Function Declaration)

```
func main(void)
{
 int a;
 int b;
 scan(a);
 scan(b);
 myAdd(a,b);
}
```

```
Valid Declaration
Valid Declaration
Valid function call
Valid function call
Valid function call
Valid function definition
#END-OF-FILE#
-------Report:-------
Correct Words: 30
Correct Expressions: 6
Incorrect Words: 0
Incorrect Expressions: 0
```

```
func myAdd(int num1, int num2){
    int sum;
    sum = a + b;
    print("Sum is: ", sum);
}
```

```
Valid Declaration
Valid Declaration
Valid Declaration
Valid assignment
Valid function call
Valid function definition
#END-OF-FILE#
-------Report:-------
Correct Words: 27
Correct Expressions: 6
Incorrect Words: 0
Incorrect Expressions: 0
```

```
func myfunc(int a, int b)
{
    int a = 0;
    for(i = 0; i < 10; i++)
    {
        if(i == 1)
        {
            print("Correct", i);
        }
        else
        {
            print("Error");
        }
    }
    print("\n");
    while(a >= 0)
    {
        print("unend");
    }
}
```

```
Valid Declaration
Valid Declaration
Valid Declaration
Valid function call
Valid if statement
Valid function call
Valid else statement
Valid for statement
Valid function call
Valid function call
Valid while statement
Valid function definition
#END-OF-FILE#
-------Report:-------
Correct Words: 72
Correct Expressions: 14
Incorrect Words: 0
Incorrect Expressions: 0
```

Στο στάδιο αυτό πραγματοποιείται έλεγχος σε συναρτήσεις που ορίζονται από το χρήστη. Αναλύονται τα ονόματα των συναρτήσεων, τα ορίσματα που δέχονται, καθώς και οι εντολές εντός του σώματος της εκάστοτε συνάρτησης. Στο πρώτο παράδειγμα, πραγματοποιείται κλήση των συναρτήσεων scan (built-in), καθώς και της myAdd (custom συνάρτηση). Παρατηρούμε ότι ο μεταγλωττιστής εντοπίζει τις κλήσεις αυτές, δίνοντας το μήνυμα valid function call για καθεμία στο output. Στο τέλος εμφανίζει και το μήνυμα valid function declaration, καθώς έχει ολοκληρωθεί το σώμα της συνάρτησης. Ομοίως λειτουργεί και το δεύτερο παράδειγμα με τη δήλωση της myAdd. Τέλος, έχουμε τη myfunc, εντός της οποίας περιέχονται δομές ελέγχου όπως if/else, καθώς και δομές επανάληψης while/for. Για την εκάστοτε δομή που εντοπίζεται ορθά, στο output εμφανίζονται μηνύματα τύπου valid if statement, valid for statement κ.ο.κ.

3.1 Μέρος: Β-3

```
bison -d syntax_analyzer.y
syntax_analyzer.y: conflicts: 20 shift/reduce
flex lexical_analyzer.l
gcc -o syntax_analyzer syntax_analyzer.tab.c lex.yy.c
-lm
./syntax_analyzer 07_in_err.txt > 07_out_err.txt
                Bison -> PARSING FAILED with (6 syntax
error(s) and 6 warning(s)).
```

```
func 5 agg ( int a );
print(break a);
int k = a ! ;
print (len a);
int a = a @ + 1;
x1 = [1,2] , [3,4];
a = [1,2] - [3,4];
int k = "agg";
for (i = 0; i < 10; i++) {
    print(i);
    a + "agg";
}
cmp(5, "a");
int a= 1.1;
x = [1, "a"];
```

```
Valid Declaration
Warning: Number found in func declaration
Warning: Invalid Keyword found in print
Warning: Operator found in declaration
Warning: Len/cmp not used correctly
Unrecognized character(s) encountered!
Valid Declaration
        Line 5 at lexeme '@' : syntax error
                1 character(s) ignored so far
Warning: Remaining elements not assigned
Warning:Concat failed keeping first array
Valid assignment
        Line 8 at lexeme '"agg"' : syntax error
Valid function call
        Line 11 at lexeme ';' : Invalid assignment
Valid for statement
        Line 13 at lexeme ')' : Invalid function
call
        Line 14 at lexeme ';' : Invalid
declaration type
        Line 15 at lexeme '"a"' : syntax error
#END-OF-FILE#
-------Report:-------
Correct Words: 110
Correct Expressions: 7
Incorrect Words: 1
Incorrect Expressions: 12
```

Στο παράδειγμα αυτό βλέπουμε ένα αρχείο κειμένου(είσοδος) για την αναγνώριση των προειδοποιητικών λαθών αλλά και των συντακτικών λαθών. Αρχικά αφού τρέξουμε το Makefile μας με την εντολή make θα δούμε ότι απέτυχε το parsing επειδή βρέθηκαν συντακτικά λάθη (6 συντακτικά και 6 warnings).

Στην πρώτη γραμμή του αρχείου βλέπουμε έναν ορισμό συνάρτησης όπου στο αρχείο εξόδου βλέπουμε ότι εκτυπώνει πρώτα Valid Declaration και μετά Warning. Αυτό συμβαίνει διότι πρώτα εκτυπώνει για το όρισμα της συνάρτησης που έχει οριστεί σωστά και μετά για όλη την γραμμή που βρίσκει το warning που έχουμε ορίσει.

Στην δεύτερη γραμμή έχουμε το δεύτερο warning που έχουμε ορίσει που βρήκε μέσα στην print ένα keyword εκτός του len/cmp που θα δούμε στην τέταρτη γραμμή. Άρα εκτυπώνει το μήνυμα που έχουμε βάλει στο αρχείο εξόδου. Στην τρίτη γραμμή έχουμε το int k = a ! ; όπου βρίσκει ένα operator πριν το delimiter και εκτυπώνει το μήνυμα του warning που έχουμε βάλει. Στην τέταρτη γραμμή είναι παρόμοιο warning με την δεύτερη γραμμή αλλά βρέθηκε len ή cmp άρα έχει οριστεί λάθος η εσωτερική συνάρτηση.

Στην γραμμή 5 έχουμε το πρώτο συντακτικό λάθος βρέθηκε στο declaration άγνωστος χαρακτήρας. Στην 6 γραμμή διαχειριζόμαστε άλλο ένα warning. Αυτή την φορά αν δώσει παραπάνω πίνακες στην δεξιά μεριά από ότι είναι τα ορίσματα στην αριστερή. Εδώ βλέπουμε ότι με ένα όρισμα έχουν δοθεί δυο πίνακες με αποτέλεσμα να βγάλει το warning που έχουμε βάλει.

Στην έβδομη γραμμή ένα παρόμοιο warning με την προηγούμενη αλλά όταν γίνεται concat μεταξύ δύο πινάκων να κρατάει το πρώτο πίνακα. Ξαναγυρίζουμε σε ένα συντακτικό λάθος στην γραμμή 8 όπου εδώ βλέπουμε ότι έβγαλε λάθος όταν σε έναν ακέραιο βάλεις μία συμβολοσειρά. Από την γραμμή 9 έως και την γραμμή 12 ορίζεται μια for loop όπου αρχικά ορίζεται σωστά και υπάρχει σωστά η συνάρτηση print αλλά στην γραμμή 11 γίνεται μια απλή πρόσθεση μεταβλητής με συμβολοσειρά όπου πετάει και το συντακτικό λάθος ότι είναι λάθος assignment. Μετά το for loop στην γραμμή 13 ορίζεται λάθος η cmp γιατί δεν γίνεται να έχει ακέραιο και συμβολοσειρά οπότε πετάει μήνυμα συντακτικού λάθους. Αντίστοιχα και στην δέκατη τέταρτη γραμμή όπου πετάει συντακτικό λάθος διότι δεν μπορείς να βάλεις σε έναν ακέραιο πραγματική τιμή.

Τέλος στην 15 και τελευταία γραμμή του αρχείου εισόδου έχουμε συντακτικό λάθος διότι δεν γίνεται σε έναν πίνακα να υπάρχουν δύο διαφορετικού τύπου μεταβλητές οπότε μας πετάει συντακτικό λάθος. Μετά το τέλος του αρχείου εισόδου στο αρχείο εξόδου εμφανίζεται ένας πίνακας ο οποίος μας δείχνει τον αριθμό των σωστών λέξεων, των σωστών εκφράσεων αλλά και των λάθος λέξεων και εκφράσεων

4. Ανάλυση προβλημάτων/ελλείψεων

Γενικά, το πρόγραμμα μεταγλωττίζεται και τρέχει με τα αρχεία εισόδου που φτιάξαμε κανονικά. Τρέχει επίσης και από το stdin αν δεν δοθεί όνομα αρχείο στην γραμμή εντολών. Στην μεταγλώττιση απλώς, υπάρχουν προειδοποιητικά σφάλματα (warnings) που αφορούν σε shift/reduce conflicts. Η shift/reduce είναι μια κατάσταση όπου ο αναλυτής μπορεί είτε να μετατοπίσει (shift) το επόμενο σύμβολο εισόδου είτε να μειώσει τη στοίβα (reduce) χρησιμοποιώντας έναν κανόνα παραγωγής. Αυτό γίνεται δηλαδή σε κάποιους γραμματικούς κανόνες μας που είναι λογικά ίδιοι μεταξύ τους σε κάποια σημεία και δεν καταφέραμε να τις υλοποιήσουμε χωρίς να εμφανίζονται αυτά τα warnings.

Γενικά, έχουμε υλοποιήσει τις δομές που περιγράφονται από το κεφάλαιο 2.1 του έγγραφου που αναλύει την συντακτική ανάλυση της Uni-C. Δεν έχουμε πραγματοποιήσει τους κανόνες σύνδεσης φυσικών γραμμών (κεφάλαιο 2.1.2.1). Ειδικότερα, στην γραμματική declaration (ανάθεση τιμών) για την διατύπωση π.χ int a = a + 1; δεν κάνουμε έλεγχο αν είναι του ίδιου τύπου οπότε μπορεί να γραφτεί int a = a + 1.1;. Ειδικότερα, π.χ για την κλήση συνάρτησης ορισμένη από τον χρήστη δεν έχουμε κάνει έλεγχο αν έχει δηλωθεί πιο πριν. Και γενικά σε πολλές γραμματικές δεν έχουμε κάνει τόσο εξαντλητικό έλεγχο για τον τύπο και γενικά για την σημασιολογική ανάλυση, αλλά έχουμε επικεντρωθεί στην γραμματική. Στον περισσότερο βαθμό έχουμε επικεντρωθεί στα παραδείγματα που έχουν δοθεί στο έγγραφο της Uni-C και δεν έχουμε επεκτείνει τις γραμματικές π.χ για όλα τα keywords όπως return στο τέλος της συνάρτησης, struct, case κτλ. Επίσης, στο σώμα της συνάρτησης έχουμε καταφέρει μόνο να γράφονται οι γραμματικές declaration, assignment, func_call, for όσες φορές θέλουμε ενώ οι δομές if while μπορούν μόνο μια φορά να γράφονται μέσα σε brackets. Αυτό πιθανόν οφείλεται στον τρόπο που έχουμε ορίσει στους γραμματικούς κανόνες για την λεκτική μονάδα END (\n) στην γραμματική all ή πιθανόν στην γενική γραμματική της if_while που είναι ενσωματωμένες μαζί. Γενικώς δε δεν καταφέραμε να τα κάνουμε να δουλεψει και για αυτό το παράδειγμά μας στην εξαντλητικό έλεγχο είναι συγκεκριμένο. Οι δομές if, while δουλεύουν όμως από μόνες τους όταν δεν είναι μέσα σε body από bracket σε ξεχωριστά αρχεία (που δεν συμπεριλάβαμε).

Επομένως, ότι άλλο λάθος υπάρχει είναι από απερισκεψία μας και λόγω της πολυπλοκότητας όλων των γραμματικών στο αρχείο. Στους εξαντλητικούς ελέγχους, ουσιαστικά αναλύσαμε τι βγαίνει σωστό αλλά γενικά όπως είναι ορισμένες οι γραμματικές μπορεί να υπάρχουν και συντακτικά σωστά που δεν προβλέψαμε λόγω των συνδυασμών των διάφορων γραμματικών και γενικά στην πολυπλοκότητα του προγράμματος.

5. Επίλογος

5.1 Τελικά Σχόλια

Η ανάλυση των παραπάνω ενοτήτων της λεκτικής, συντακτικής καθώς και σημασιολογικής ανάλυσης είναι θεμελιώδη όσον αφορά την κατανόηση του μεταγλωττιστή που έχει κατασκευαστεί. Τα προγράμματα σε συνδυασμό μεταξύ τους αποτελούν τον λεγόμενο μεταγλωττιστή. Στην περίπτωση μας, έχει υλοποιηθεί μία σχετικά απλή μορφή ενός μεταγλωττιστή, στην οποία ο μεταγλωττιστής είναι ικανός να πραγματοποιήσει. Είναι ικανός να αναγνωρίσει λεκτικές μονάδες όπως αριθμούς, συμβολοσειρές, τελεστές, αναγνωριστικά και άλλα με χρήση του lexical analyzer (lexer). Διεξάγει τους απαραίτητους ελέγχους έτσι ώστε οι λεκτικές μονάδες να είναι (συντακτικά) σωστά συνδεδεμένες μεταξύ τους. Ελέγχει για αριθμητικές, λογικές, συγκριτικές εκφράσεις, εκφράσεις δήλωσης μεταβλητών, κλήσης/δήλωσης συναρτήσεων κτλ. Επιπλέον περιέχει ελέγχους για δομές ελέγχου όπως if, καθώς και για δομές επανάληψης όπως while/for. Κατά τη διαδικασία της μεταγλώττισης, το πρόγραμμα μπορεί να εντοπίσει τυχόν λάθη, είτε λεκτικά είτε συντακτικά, τα οποία τυπώνει έπειτα από τη διαδικασία της μεταγλώττισης σε μορφή αναφοράς.

5.2 Υποσημείωση/Διόρθωση για Μέρος Α-3

Στο μέρος Α-3 στον κώδικα Flex, έγινε ένα συντακτικό λάθος στην διατύπωση της κανονικής έκφρασης για την λεκτική μονάδα που αφορά τα Strings. Στο flex αρχείο στην γραμμή 19 είναι διατυπωμένη ως εξής:

```
STRINGS "(\\[\\n\"]|[^\n\\\"])*"
```

Αν και σημασιολογικά αυτή είναι η κανονική έκφραση για τα String (που περιγράψαμε και ελέγξαμε στο μέρος Α-2) στον Flex έπρεπε να διατυπωθεί ως εξής:

```
STRINGS \"(\\[\\n\"]|[^\n\\\"])*\"
```

Δηλαδή, η μόνο διαφορά είναι στα εξωτερικά " που έπρεπε να γραφτούν \" για να αναγνωρίζει τον χαρακτήρα. Κάνοντας αυτήν την αλλαγή τα Strings αναγνωρίζονται κανονικά από τον flex και αυτήν την αλλαγή την έχουμε κάνει στην μορφή του lexical_analyzer.l για το μέρος Β.

6. Ανάλυση Αρμοδιοτήτων

Το μέλος της ομάδας «Εντερίσα Γκιόζι» που βοήθησε στην δημιουργία Α μέρους (Α-2, Α-3) δεν συμμετείχε καθόλου σε όλη την διάρκεια υλοποίησης της εργασίας μέρους Β (Β-2 ,Β-3) και αποχώρησε οικειοθελώς πριν την έναρξη της υλοποίησης. Για αυτόν τον λόγο, παρακάτω αναγράφονται οι αρμοδιότητες για τα υπόλοιπα τέσσερα μέλη.

3.1 Γενικές Αρμοδιότητες

		Κοντούλης Δημήτριος (21390095)	**Μεντζέλος Άγγελος Κωνσταντίνος (21390132)**	**Βάρσου Ευφροσύνη (21390021)**	**Αλεξόπουλος Λεωνίδας (2139006)**
Υλοποίηση κώδικα (Γραμματικών)	Δηλώσεις Μεταβλητών			√ Πίνακες, Ιδίου τύπου αναθέσεις	√ Απλή δήλωση
	Built-in συναρτήσεις	√ cmp, scan		√ len, print	
	Δήλωση συναρτήσεων χρήστη	√ Δήλωση συνάρτησης	√ Κλήση συνάρτησης		
	Δήλωση απλών εκφράσεων		√ Αριθμητικές Εκφράσεις, Συγκρίσεις, Συνένωση πινάκων	√ Αναθέσεις τιμών σε αναγνωριστικά (και ομαδοποιήση)	
	Σύνθετες δηλώσεις	√ For δομή	√ While δομή	√ If δομή	
	Καταμέτρηση λέξεων				√ Σωστών και Λανθασμένων
	Καταμέτρηση εκφράσεων		√ Λανθασμένων	√ Σωστών	

	Τύπωμα ανάλυσης			√	
	Είσοδος από αρχείο		√ Διόρθωση	√ Υλοποίηση	
Διαχείριση Προειδοποιητικών λαθών		√	√	√	
Δημιουργία αρχείων εισόδου		√	√	√	√
Έλεγχος εξόδου		√	√		
Σχολιασμός κώδικα		√ Αναλυτικός		√ Επιμέρους	

3.2 Υλοποίηση Word

		Κοντούλης Δημήτριος (21390095)	**Μεντζέλος Άγγελος Κωνσταντίνος (21390132)**	**Βάρσου Ευφροσύνη (21390021)**	**Αλεξόπουλος Λεωνίδας (2139006)**
1. Εισαγωγή				√	
2. Τεκμηρίωση Γραμματικής	2.1.1Λεκτική Ανάλυση			√	
	2.1.2 Συντακτική Ανάλυση			√	
	2.1.3 Σημασιολογική Ανάλυση	√			
	2.2 Διαχείριση Προειδοποιητικών λαθών		√		
3. Παρουσίαση Εξαντλητικών Ελέγχων		√	√	√	√
4. Ανάλυση προβλημάτων /ελλείψεων				√	
5. Επίλογος		√			
6. Ανάλυση αρμοδιοτήτων				√	