Blue text on a black background

Description automatically generated

***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: 18/5/2024***

***Τμήμα Β2 - Ομάδα 2***

*ΚΟΝΤΟΥΛΗΣ ΔΗΜΗΤΡΙΟΣ 21390095*

*ΜΕΝΤΖΕΛΟΣ ΑΓΓΕΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ 21390132*

*ΒΑΡΣΟΥ ΕΥΦΡΟΣΥΝΗ 21390021*

*ΑΛΕΞΟΠΟΥΛΟΣ ΛΕΩΝΙΔΑΣ 21390006*

Περιεχόμενα

[1. Εισαγωγή 3](#_Toc169361753)

[2. Τεκμηρίωση Γραμματικής 3](#_Toc169361754)

[2.1 Λεκτική Ανάλυση 3](#_Toc169361755)

[2.2 Συντακτική Ανάλυση 3](#_Toc169361756)

[2.3 Σημασιολογική Ανάλυση 3](#_Toc169361757)

[3. Παρουσίαση Εξαντλητικών ελέγχων 3](#_Toc169361758)

[4. Ανάλυση προβλημάτων/ελλείψεων 3](#_Toc169361759)

[4.1 Μέρος Β-2 3](#_Toc169361760)

[4.2 Μέρος Β-3 3](#_Toc169361761)

[5. Επίλογος 3](#_Toc169361762)

[5.1 Τελικά Σχόλια: 3](#_Toc169361763)

[Υποσημείωση/Διόρθωση για Μέρος Α-3 3](#_Toc169361764)

[6. Ανάλυση Αρμοδιοτήτων 4](#_Toc169361765)

[3.1 Γενικές Αρμοδιότητες 4](#_Toc169361766)

[3.2 Υλοποίηση Word 4](#_Toc169361767)

# 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 τυπών. Δηλαδή αν μία μεταβλητή έχει δηλωθεί ως ακέραιος (int), τότε δεν είναι εφικτό να της ανατεθεί τιμή η οποία είναι πραγματικός αριθμός (float), ή κάποιο άλλος τύπος όπως string.

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.

Αρχικά, στο label με όνομα keyword\_val, υπάρχει ένα set εντολών, οι οποίες επιμέρους εντολές, δέχονται κάποιες δεσμευμένες λέξεις και τις αντιστοιχούν σε αριθμητικές τιμές. Να σημειωθεί ότι το $1 αφορά την τιμή που δέχεται το σύμβολο keyword και το $$ αφορά την τιμή που δέχεται το σύμβολο keyword\_val. Η ανάθεση των λέξεων αυτών σε τιμές είναι ιδιαίτερα σημαντική για την σημασιολογική ανάλυση, καθώς μας επιτρέπουν να πραγματοποιησούμε ελέγχους για διάφορες δομές, όπως θα δούμε και παρακάτω. Με την ίδια λογική, κατασκευάζουμε και τις διαφορετικές τιμές για κάθε πιθανό operator, στο label oper\_val. Προχορώντας περαιτέρω στον κώδικα, παρατηρούμε το σύμβολο expr, μέσα στο οποίο περιέχονται οι πράξεις που εκτελεί κάθε operator. Σε περίπτωση που ο αριθμός του συμβόλου num oper\_val num είναι 1, πρόκεται για τον τελεστή “+”, ο οποίος πραγματοποιεί μία πρόσθεση μεταξύ των δύο num values. Αντίστοιχα δουλεύουν οι υπόλοιποι τελεστές στον ίδιο γραμματικό κανόνα, καθώς και οι υπόλοιποι γραμματικοί κανόνες γενικότερα στο ίδιο label (expr).

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.

Παραπάνω πχ είδαμε για το declaration και assignment μεταβλητών. Περαιτέρω, το πρόγραμμα περιέχει ελέγχους για δομές if, while. Στις δομές επανάληψης, όπως είναι η while, πρέπει να ελέγχουμε τις τιμές αυτές, έτσι ώστε έχουμε δομή while η οποία κλείνει με ένα κομμάτι else. Αυτή η περίπτωση αφορά αποκλειστικά τη δομή if, για αυτό και γίνονται οι απαραίτητοι έλεγχοι. Ας δούμε και τις περιπτώσεις των built-in συναρτήσεων print και len. Και οι δύο συναρτήσεις 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 πρόγραμμα μας, η συνάρτηση len αναπαρίσταται από τον αριθμό 4 και η συνάρτηση print από τον αριθμό 6. Στο τμήμα scan\_len\_print πραγματοποιούνται οι απαραίτητοι σημασιολογικοί έλεγχοι των δύο αυτών συναρτήσεων, με χρήση των συμβόλων $1 και $3, που αφορούν το keyword ($1) και το εκάστοτε είδος του array ($3). Να σημειωθεί ότι το τμήμα scan\_len\_print, αφορά την χρήση των συναρτήσεων αυτών με μόνο ένα όρισμα. Επιπρόσθετα, εντός της print, μπορεί να δοθεί το όνομα ενός πίνακα, συνοδευμένο από ένα index, με το οποίο μπορούμε να εκτυπώσουμε ένα συγκεκριμένο κομμάτι του πίνακα. Αυτό με τη χρήση της len δεν είναι εφικτό, καθώς πρέπει να συντάξουμε την εντολή ως len(arr). Ενώ με τη print, μπορούμε να γράψουμε print(arr[0]), επιτρέποντας της έτσι να τυπώσει το πρώτο στοιχείου που περιέχεται στον πίνακα που δέχτηκε ως όρισμα.

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)

ΑΓΓΕΛΟΣ

# 3. Παρουσίαση Εξαντλητικών ελέγχων

ΑΓΓΕΛΟΣ Β-3 και θα μας πει άμα θέλει βοήθεια

ΚΟΝΤΟΥΛΗΣ 3-4 αρχεία από Β-2 και ότι πει ο άγγελος για β-3

ΦΡΟΣΩ 3-4 αρχεία από Β-2 και ότι πει ο άγγελος για β-3

# 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.

## 4.1 Μέρος Β-2

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 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 και οι δομές if while και for 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 και δεν υπάρχει ο γραμματικός κανόνας για να είναι σωστό αλλά δεν καταφέραμε να τα κάνουμε να δουλεύει. Οι δομές if, while, for δουλεύουν όμως από μόνες τους όταν δεν είναι μέσα σε body από bracket σε ξεχωριστά αρχεία.

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.

## 4.2 Μέρος Β-3

ΑΓΓΕΛΟΣ

# 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.

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 Λεκτική Ανάλυση |  | **√** |  |  |
| 2.2 Συντακτική Ανάλυση |  |  | **√** |  |
| 2.3 Σημασιολογική Ανάλυση | **√** |  |  |  |
| 3. Παρουσίαση Εξαντλητικών Ελέγχων | | **√** | **√** | **√** | **√** |
| 4. Ανάλυση προβλημάτων /ελλείψεων | |  | **√ Β-3** | **√ Β-2** |  |
| 5. Επίλογος | | **√** |  |  |  |
| 6. Ανάλυση αρμοδιοτήτων | |  |  | **√** |  |